ФГБОУ ВО «Московский государственный психолого-педагогический университет»

На правах рукописи

Попков Сергей Игоревич

МОДЕЛИРОВАНИЕ ПОВЕДЕНИЯ ВЕРОЯТНОСТНЫХ МНОГОАГЕНТНЫХ СИСТЕМ С ДЕЦЕНТРАЛИЗОВАННОЙ АРХИТЕКТУРОЙ

05.13.18 – Математическое моделирование, численные методы и комплексы программ

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель
Доктор технических наук, профессор
Куравский Лев Семенович

Москва, 2019

Оглавление

Введение

Общая характеристика работы

Проблемы исследования коллективного поведения и организации группового управления привлекают внимание исследователей с 30-х годов 20-го века [1-4]. Их значимость значительно возросла в последние годы в связи с актуальностью задач управления коллективом роботов, включая группы беспилотных летательных аппаратов и других мобильных систем. Особенно трудными являются задачи управления группами подвижных объектов, которые должны координировать свое поведение в пространстве и кооперироваться для достижения заданного результата. В настоящее время исследования и разработки в этой области ведутся во многих странах с привлечением большого количества специалистов. Под многоагентными системами далее понимаются системы, образованные совокупностью взаимодействующих интеллектуальных агентов. Класс многоагентных систем с децентрализованной архитектурой, в отличие от централизованной, определяет агентов в рамках системы как равноценных и способных, при необходимости, действовать автономно согласно определенной цели.

За прошедшее время подходы к решению подобных задач значительно изменились, требуя, в частности, новых способов формализации, математических моделей и алгоритмов адаптивного управления поведением прикладных распределённых многоагентных систем. Концепции, базовые понятия, принципы построения, перспективы практического применения и другие аспекты создания многоагентных систем рассмотрены во многих современных работах, в том числе Вяткина А.Ю., Смирнова Д.В., Кочетова И.А., Городецкого В.И., Карсаева О.В., Самойлова В.В., Пантелеева А.В., Скавинской Д.В., Серебрякова С.В., Бухвалова О.Л., Скобелева П.О. и других

исследователей [5-16]. Однако математический аппарат и программные средства, приемлемые для управления поведением составляющих систему агентов в реальном времени и оперативной оценки ресурсов, необходимых для решения прикладной задачи, к настоящему времени в полном объёме ещё не созданы. Отсутствует пригодное к практическому применению математическое и программное обеспечение оценки уровня подготовки и обучения операторов многоагентных систем на специализированных тренажёрах.

Поведение ряда технических, диагностических и других систем часто удобно описывать, используя специально разработанные параметрические математические модели в форме систем обыкновенных дифференциальных уравнений. Подобная форма представления применялась для ряда моделей, среди которых можно выделить модели усталостного разрушения, адаптивного тестирования, согласованного управления подвижными объектами и других [17-21].

Несмотря на то, что скорость оптимизации с применением описанного подхода, как правило, близка к приемлемой, существует целый ряд задач, для которых одним из требований является мгновенное принятие решений. Задачи группового управления в реальном времени относятся к классу подобных задач. В частности, при достаточно большом количестве параметров необходимо обеспечить адаптивность алгоритма оптимизации в отношении заведомо установленных параметров алгоритма, влияющих на скорость его работы, таким образом, чтобы увеличить скорость сходимости без потери качества получаемого в результате работы метода значения критерия.

Кроме того, необходима реализация адекватной математической модели в целом, которая бы позволила не только осуществлять процесс управления агентами, но и помогать эффективно прогнозировать исход боя за счет специального набора инструментов, а также предпринимать наиболее верные стратегические шаги лицу, принимающему решение, исходя из имеющихся ресурсов – количества агентов, снаряжения, специфических характеристик агентов, временных затрат и т.п.

Применение такой модели на практике может быть полезно не только при решении задач поиска оптимальной стратегии, но и для повышения квалификации и определения уровня компетентности операторов сложных систем, связанных с предметной областью моделируемой системы. Применяемые в настоящее время в РФ боевые комплексы управляются операторами, что делает их неоправданно уязвимыми и создаёт проблемы при оперативном принятии решений в боевой обстановке, существенно понижая эффективность управления. Полная автоматизация обеспечивает повышение эффективности использования комплексов данного типа, в частности, за счёт оптимизации их поведения и увеличения скорости принятия решений. Однако в ситуациях, когда полная автоматизация, в силу ряда причин, не представляется возможной, а присутствие человека, напротив, необходимо и неизбежно, существует потребность в наличии компонента человеко-машинного взаимодействия, который позволил бы вести обучение и определять уровень компетентности потенциальных операторов многоагентной системы для управления средствами поражения цели, и реализация такого рода обучающего процесса с практической точки зрения полезна тем, что позволила бы избежать выхода из строя реального дорогостоящего оборудования во время тренировки за счет внедрения

соответствующего программного комплекса, контролирующего правдоподобную имитацию ведения боевых действий.

Были рассмотрены аналогичные системы, работа которых могла бы обеспечить функционирование описанного процесса обучения. Для некоторых из них характерно наличие математической модели, описывающей адаптивное обучение, однако ограничения реализации в рамках конкретной предметной области не позволяют организовать процесс обучения для оператора сложных систем в общем виде. Для других систем не предусмотрено наличие специализированной математической модели для оценки уровня обучения по заданным критериям, несмотря на адаптацию к предметной области сложных систем. Таким образом, возникает потребность в разработке и программной реализации виртуального адаптивного тренажера, учитывающего недостатки исследованных подходов к решению поставленной задачи обучения.

Описанные выше практические проблемы моделирования и анализа коллективного поведения, а также применения группового управления позволяют говорить о необходимости:

— разработки новых методов математического моделирования поведения многоагентных систем, позволяющих эффективно прогнозировать поведение этих систем на основе вероятностных оценок в реальном времени;
— разработки численных методов, учитывающих требования к современным шаблонам проектирования программных продуктов и позволяющих оптимизировать вычислительные процедуры, прогнозирующие поведение многоагентных систем;

— разработки кроссплатформенного комплекса программ для имитационного моделирования, обеспечивающего оценку уровня подготовки оператора сложной многоагентной системы и реализующего принципы адаптивного обучения на специализированных тренажёрах.

Актуальность темы диссертации обусловлена необходимостью создания новых подходов к решению задачи группового управления многоагентными системами, обеспечивающих прогнозирование моделируемой ситуации и принятие решений на основе количественных критериев, а также оценку уровня подготовки и обучение операторов, работающих с этими системами.

Цель работы: разработка математической модели управления поведением многоагентных систем и реализация на её основе комплекса программ для прогнозирования такого поведения и оценки уровня подготовки и обучения операторов специализированных тренажёров.

Для достижения цели в процессе исследования решены следующие **задачи**:

— создания математической модели для исследования поведения частного класса многоагентных систем;
— создания математической модели и метода прогнозирования, обеспечивающих оценку ресурсов, необходимых для решения задачи, на основе количественных критериев;
— проведения вычислительных экспериментов, необходимых для оценки параметров прогнозирования;

— идентификации параметров модели исследуемой прикладной многоагентной системы;
— создания на основе разработанных моделей и методов комплексов программ для прогнозирования поведения исследуемой прикладной многоагентной системы и оценки уровня подготовки и адаптивного обучения её операторов.

Методологические основы и методы исследования: для решения поставленных задач использовались модели теории случайных процессов, методы анализа данных и оптимизации, а также численные методы.

На защиту выносятся следующие научные результаты:

— математическая модель и алгоритм поведения прикладной многоагентной системы исследуемого класса в реальном времени;
— математическая модель и метод прогнозирования, обеспечивающие оперативную оценку ресурсов, необходимых для решения задачи, на основе количественных критериев;
— адаптивный численный метод идентификации параметров модели прикладной многоагентной системы;
— комплексы программ для прогнозирования поведения прикладной многоагентной системы и оценки уровня подготовки и адаптивного обучения её операторов.

Научная новизна заключается:

— в математической модели и алгоритме поведения прикладной многоагентной системы;
— в математической модели и методе прогнозирования, обеспечивающих оперативную оценку ресурсов, необходимых для решения задачи;

— в адаптивном численном методе идентификации параметров прикладной многоагентной системы;
— в концепциях, лежащих в основе разработанных комплексов программ.

Практическая значимость диссертационной работы заключается в возможности:

— создания на основе разработанных математических моделей и алгоритмов прикладных многоагентных систем с полностью или частично автоматизированным групповым управлением в реальном времени;
— вычисления на основе разработанных методов прогнозирования лицом, принимающим решения, оценок ресурсов, необходимых для решения поставленной задачи;
— оценок уровня подготовки и адаптивного обучения на специализированных тренажёрах операторов многоагентных систем с использованием созданных комплексов программ.

Достоверность результатов работы подтверждается:

— оценками военных экспертов Всероссийского конкурса Министерства обороны РФ 2018 года по поиску в интересах Вооруженных Сил Российской Федерации научно-исследовательских работ граждан Российской Федерации,
— оценками экспертов Всероссийского межотраслевого молодёжного конкурса научно-технических работ и проектов «Молодёжь и будущее авиации и космонавтики»,
— сопоставлением результатов имитационного моделирования с эмпирическими данными,

— вычислительными экспериментами, подтвердившими эффективность разработанного численного метода идентификации параметров прикладной многоагентной системы.

Апробация: Работа стала победителем финального этапа Всероссийского межотраслевого молодёжного конкурса научно-технических работ и проектов «Молодёжь и будущее авиации и космонавтики» в 2018 году в номинации "Математические методы в аэрокосмической науке и технике" и заняла II место на Всероссийском конкурсе Министерства обороны РФ 2018 года по поиску в интересах Вооруженных Сил Российской Федерации научно-исследовательских работ граждан Российской Федерации. Теоретические и практические результаты работы были представлены на Всероссийских научных конференциях «Нейрокомпьютеры и их применение» в 2016-2019 годах (отмечены дипломами за лучший научный доклад), Всероссийской выставке научно-технического творчества молодежи «НТТМ-2015» (отмечены дипломом «НТТМ-2015»), а также на научных семинарах в Военной академии РВСН имени Петра Великого и Главном научно-исследовательском испытательном центре робототехники Министерства обороны РФ. Работа прошла регистрацию в Роспатенте [22].

Современное состояние проблемы

Несмотря на многообразие сфер применения и способов реализации, все многоагентные системы — это системы, образованные совокупностью взаимодействующих интеллектуальных агентов. Под термином «интеллектуальный агент» (далее – «агент») понимается процесс, получающий информацию в виде данных о совокупности других управляемых процессов и способный влиять через управление этими процессами, способствуя достижению поставленной цели. Под окружающим миром понимается совокупность целей, с которыми взаимодействует агент, самих агентов и окружения, задаваемого моделируемой системой.

Набор средств получения информации и восприятия окружающего мира называется сенсорами агента, а совокупность механизмов, осуществляющих воздействие на окружающий мир — актуаторами агента.

Особенности практического применения рассматриваемых систем требуют, чтобы каждый интеллектуальный агент обладал:

- умением выполнять задачи в сложном окружении без постоянной поддержки извне (автономностью);
- способностью улучшать качество выполняемой работы на основе приобретенного опыта (адаптивностью);
- способностью к организации деятельности в соответствии с алгоритмом функционирования;
- собственной целевой функцией.

При этом иерархия взаимодействия среди агентов в системах с децентрализованной архитектурой должна отсутствовать - не допускается

существование агентов, управляющих всей системой. В таких многоагентных системах, несмотря на децентрализацию, может находить свое проявление феномен самоорганизации системы, а также иметь место синергетический эффект, когда сравнительно простое индивидуальное поведение агентов может приводить к появлению сложной стратегии поведения у класса этих агентов как совокупности — этот принцип лежит в основе так называемого «роевого интеллекта».

Многоагентные системы берут свое начало в концепции многопоточного программирования, расширяя ее до универсальной абстракции. Такой взгляд на многоагентные системы позволяет масштабировать системы искусственного интеллекта, распараллеливая решаемые задачи на уровне модели. Например, если стоит задача распознавания образа, соответствующий алгоритм применяется относительно совокупности участков распознаваемого изображения, причем за каждый участок отвечает независимый агент, и, в общем случае, алгоритм, задающий действия агента, может адаптироваться под особенности конкретного участка, оптимизируя процесс распознавания в целом.

Многоагентные системы применяются в ряде областей для решения разных задач, среди них:

1) Анализ поведения синергетически связанных объектов (управление группой спутников в космосе, повышение уровня безопасности маневров воздушных судов и т.д.)
2) Военное дело (РТК, беспилотные летательные аппараты, «рой» роботов)
3) Исследования в прикладных областях (наномедицина, онлайн-торговля, логистика и т. д.)

4) Исследование социальных и биологических систем
5) Прогнозирование и моделирование процессов из соответствующих предметных областей (социальных структур, ликвидации ЧС и т.д.)
6) Подготовка кадров для управления комплексными системами (задачи тренировки и определения уровня компетентности)
7) Организация и оптимизация процесса поиска информации

Существуют два способа разработки многоагентных систем: первый — с помощью агентных платформ, предназначенных для реализации произвольных агентов в некоторой промежуточной универсальной среде; второй — с помощью среды разработки, когда агенты задаются непосредственно кодом алгоритма, а окружающий мир задается совокупностью агентов.

Генерация многоагентных систем с помощью среды разработки, как правило, не применяется на практике. Одна из основных причин, по которым такой способ зачастую неприменим к серьезным практическим задачам – нарушение принципа целостности системы и независимости среды от действующих в ней агентов. Хотя многоагентная система и должна включать для своей работоспособности, по крайней мере, одного активного агента, объективная реальность в рамках имитируемого системой процесса не должна зависеть от ее восприятия конкретными экземплярами агентов. Поэтому для создания корректно работающей и непротиворечивой многоагентной системы требуется разделять окружение, задаваемое агентной платформой, и логику взаимодействия агентов в заданной системой модели.

Существует множество уже разработанных платформ, некоторые из которых выполняют слишком узкую задачу, некоторые же в настоящее время

не поддерживаются. Ниже будут рассмотрены наиболее известные, развитые и актуальные агентные платформы.

JADE (Java Agent DEvelopment framework) — реализованный полностью на языке Java, данный фрейворк упрощает реализацию многоагентных систем и коммуникацию между агентами за счет таких технологий, как парадигма асинхронной передачи сообщений между агентами на базе peer-to-peer и нативной кроссплатформенной поддержке распределенных вычислений между компьютерами (позволяя, таким образом, использовать независимую вычислительную машину в качестве агента, в том числе, телефоны, планшеты и прочие устройства на базе Android). Распространяется по свободной лицензии LGPL.

Cougaar — изобретение DARPA (US Defense Advanced Research Projects Agency), разработанное специально для исследования потенциала многоагентных систем для задач военной логистики. Основной задачей было разработать детальный план логистики для вооруженных сил США за 180 дней, а также контроль над его выполнением с возможностью нерегулярного внесения изменений в план во время его выполнения. Программная архитектура этой системы была разработана с учетом применения в комплексных распределенных высоконагруженных приложениях с высокими требованиями к безопасности и отказоустойчивости (система должна сохранять работоспособность при отказе 45% инфраструктуры и поддерживающего оборудования с минимальным ограничением спектра допустимых операций и уровня производительности). Реализована на языке Java, распространяется по свободной лицензии на основе BSD.

Jason — платформа, позволяющая строить многоагентные системы, основанные на межъязыковом взаимодействии двух языков: императивного

объектно-ориентированного языка программирования Java для построения окружения и диалекте декларативного агенто-ориентированного языка программирования AgentSpeak для организации расширенного взаимодействия между агентами в заданном окружении (данный язык активно используется в соревнованиях по многоагентному программированию MAPC). Реализована на языке Java, распространяется по свободной лицензии LGPL.

Одна из основных проблем, ограничивающих применение существующих многоагентных платформ — высокий уровень требований к ресурсам. Cougaar изначально разрабатывался под комплексные, развернутые проекты с большим количеством серверов. Язык Java, в силу необходимости взаимодействия с виртуальной машиной, также не позволяет достичь высокого уровня производительности и не позволяет выполнять код на встраиваемых вычислительных платформах, которые, по тем или иным причинам, не могут позволить себе поддержку Java (например, встраиваемые системы реального времени).

Данная работа представляет собой попытку создать необходимый математический аппарат для частного класса многоагентных систем, область практического применения которых связана с групповым управлением «роя» агентов. В ней рассмотрена модель поведения прикладной системы, представляющей игровое взаимодействие множества агентов и цели. Поведение агентов является недетерминированным и, поэтому, непредсказуемым с точки зрения цели. Система допускает как согласованное, так и автономное поведение агентов, в зависимости от того, получают или нет агенты информацию о наличии и положении других работоспособных агентов. Поведение агента детализируется алгоритмом, который

предусматривает идентификацию параметров вероятностной модели с использованием максимизируемых целевых функций, выражающих групповую и индивидуальную вероятности поражения цели.

Для того, чтобы преодолеть ограничения и недостатки, присущие рассмотренным выше агентным платформам, необходимо разработать собственную среду – независимую агентную платформу, разработанную на базе компилируемого языка программирования, чтобы избежать ресурсных затрат времени выполнения. Кроме того, язык должен поддерживать нативную кроссплатформенность, чтобы обеспечить работу многоагентной системы при разном наборе параметров, характеризующих среду выполнения, таких как разрядность, архитектура, тип и вид операционной системы.

Математическая модель в основе многоагентной системы должна обеспечивать возможность автономного и коллективного взаимодействия интеллектуальных агентов без необходимости в осуществлении физической связи между отдельными агентами. Агенты не могут отдавать друг другу команды, чтобы осуществить взаимодействие – все необходимые данные поступают из среды выполнения, обусловленной моделью, что упрощает архитектуру системы, не ограничивая ее функциональные свойства.

Для решения задач этого класса принято использовать современные методы, относящиеся к сфере машинного обучения [23-34]. Среди таких методов наиболее распространенными являются деревья решений (лес решений), глубинное обучение с подкреплением и семейство моделей на базе генетических алгоритмов.

Далее будет рассмотрен каждый из этих методов, проведен анализ его достоинств и недостатков с точки зрения решаемой задачи.

Модели на базе деревьев решений

Деревья решений применяются для прогнозирования ситуаций, подразумевая, что у каждой моделируемой ситуации может быть несколько вариантов развития событий. Чаще всего дерево строится нисходящим, где главная, начальная вершина — ключевая проблема или исходная анализируемая ситуация. Совокупность дуг, исходящих из каждой вершины, в дереве решений определяет все возможные варианты развития событий, исходящие из возникшей ситуации, задаваемой вершиной. Каждой дуге соответствует некоторая вероятностная характеристика, вычисляемая в процессе анализа ситуации согласно некоторой формуле либо мнемоническому правилу.

Достоинства модели:

1) Надежность — природа модели исключает сбои или риск образования кластера неучтенных данных в ходе анализа ситуации и подготовки информации для принятия решений.
2) Простота интерпретации — одно из основных преимуществ деревьев в том, что их удобно представлять графически, а с помощью различных визуальных приемов (например, цветового кодирования) можно наглядно продемонстрировать наиболее адекватные поставленной задаче исходы моделируемой ситуации.
3) Отсутствие необходимости предварительной подготовки данных — в то время, как для многих других методов необходимы процедуры сравнительно сложной подготовки обрабатываемых данных

(исключение отсутствующих значений, проверка диапазона и т.п.), чтобы нивелировать их влияние на конечный результат, деревья решений могут исключить неподходящие данные в процессе анализа, облегчая, таким образом, процедуру предобработки данных.

Недостатки:

1) Модель усложняется с ростом числа возможных исходов и не применима в случае неопределенного или бесконечного числа исходов — деревья решений хорошо адаптируются к конкретно поставленной задаче в неизменяемом окружении, однако требуют просчета всех возможных исходов, что, как правило, невозможно в случае применения динамически изменяющихся вероятностных моделей.
2) Модель не подходит для сложных (составных) критериев принятия решения — дерево решений становится непригодным в сценариях моделирования, когда при изменении числа параметров системы и характера их взаимной зависимости, а также в ходе постановки новой ключевой проблемы, расширяющей исходную. В таких случаях модель требует либо полного перерасчета, что делает исходную модель бесполезной, либо полностью оказывается непригодной к использованию. В частности, деревья решений неэффективны для сопоставления эффектов коллективного и автономного воздействия агентов на определенное моделью окружение и, как следствие, не позволяют корректно решать проблемы, обусловленные поддержкой автономного и коллективного взаимодействия агентов.
3) Модель непригодна для расширения — исходя из первых двух пунктов, модель не имеет способности к масштабированию и внесению

модификаций (например, адаптации к модификации задачи на случай движущейся цели при исходно заданной статичной).

Модели на базе глубинного обучения (с подкреплением)

Глубинное обучение — это развитие идеи, лежащей в основе модели искусственных нейронных сетей, основанной на обучении исполнителя представлениям, а не конкретным алгоритмам.

Искусственные нейроны (персептроны) с помощью совокупности нелинейных фильтров для извлечения признаков с преобразованиями выстраиваются в последовательные взаимосвязанные слои таким образом, чтобы каждый слой, принимая на вход данные от предыдущего слоя, отражал все более глубокий уровень абстракции признаков. Таким образом, формируется иерархия признаков от внешнего слоя к более глубокому. На выходном слое, за счет принципа «обучения с учителем», формируется взаимосвязь заданного абстрактного понятия с изображением или другого рода информацией, предъявляемой на входном слое.

В настоящий момент существует множество различных архитектур искусственных нейронных сетей, ориентированных на решение конкретного класса прикладных задач (сверточная нейронная сеть для распознавания образов, рекуррентная нейронная сеть для синтеза и распознавания речи, и т. п.). Хотя, как правило, нейронная сеть действует на основе образцов («обучение с учителем»), существуют и другие подходы к подготовке нейронных сетей, в частности, «обучение с подкреплением». Для этой группы методов характерно отсутствие исходных данных, по которым производится обучение — в качестве «учителя» предстает сама моделируемая среда, а обучаемая структура играет роль агента, который на

основе заданной конечной цели и данных, полученных в ходе взаимодействия со средой, осуществляет обучение (накопление опыта) самостоятельно. В случае, если в роли такого агента предстает нейронная сеть, ее входной слой получает изображение окружающей среды, а конечный слой ограничен значениями, сопоставленными набору допустимых действий в среде. Взаимодействие со средой приводит к изменению изображения как результата завершенного действия и корректировке дальнейших действий в соответствии с функцией, определяющей конечную цель.

Достоинства модели:

1) Универсальность подхода — возможность применения одной и той же модели для обучения агента взаимодействию с разными окружениями, которые можно задать моделями из разных предметных областей, но с похожим набором параметров, определяющих целевую функцию.
2) Гибкость и адаптируемость к окружению — изменение характера параметров и их взаимосвязи, масштабирование или внесение модификаций в модель многоагентной системы не приводит к обесцениванию накопленного опыта у агента, а позволяет, используя его, приспособиться к новым правилам, не прибегая к математических формулам, описывающим новые особенности работы модели.

Недостатки:

1) При небольшом числе слоев модель требует избыточного числа эвристических правил для сколь-нибудь эффективного практического применения — модель не будет обладать достаточной глубиной и не сможет выделить значимое количество признаков для принятия адекватных поставленной задаче решений.

2) При адекватном числе слоев модель требует значительных ресурсов для обучения — несмотря на то, что уже обученная модель искусственной нейронной сети может запускаться и на обычных компьютерных системах, включая персональные компьютеры и смартфоны, сам процесс обучения при достаточно «глубокой» архитектуре слоев часто требует наличия нескольких распределенных кластеров высокомощных вычислительных суперкомпьютеров и может занимать продолжительное время.
3) Процесс самостоятельного порождения промежуточных признаков может привести к трудно обнаруживаемым парадоксам — применение глубинного обучения с подкреплением не допускается там, где не допускается неопределенность поведения модели (для моделей, ориентированных на военное применение, примером может служить «пацифизм», «стрельба по своим» и тому подобные формы поведения агента).

Модели на базе генетических алгоритмов

Генетический алгоритм является разновидностью принципа эволюционных вычислений, который использует процессы естественного отбора для решения задачи оптимизации, моделируя такие инструменты как наследование (скрещивание), мутацию (понимается как случайное изменение гена), отбор и кроссинговер (рекомбинация родительских генов). Задача формулируется таким образом, чтобы исходные и конечные данные можно было представить в виде вектора генов, где каждый элемент вектора представляется некоторым однородным объектом. Как правило, вектор имеет фиксированное количество элементов. Начальная популяция агентов генерируется случайным образом и выполняет задачу, исходя из имеющегося

набора генов-параметров, влияющих на поведение. Каждая особь популяции оценивается с помощью «функции приспособленности», которая определяет по некоторым заданным критериям «приспособленность» или степень качества решения агентом поставленной задачи. Из полученного множества агентов, составляющих «поколение», отбираются (например, методом рулетки) такие, у которых приспособленность выше. К ним применяются «генетические операторы» - скрещивание, мутация и другие, в зависимости от модели. Так появляются новые поколения, и процесс отбора повторяется итеративно, пока не будет достигнут некоторый заданный критерий или количество эпох обучения (поколений) не превысит заданного предела.

Достоинства модели:

1) Хорошо адаптируются под заданное окружение — в силу природы метода.
2) Обеспечивают стохастичность поведения — например, стохастичность движения агента.

Недостатки:

1) Долгая «сходимость» процесса обучения — для получения эффективного результата может пройти слишком большое количество итераций.
2) Любое изменение условий в окружении нивелирует эффект обучения — модель способна обучиться новым принципам работы, но потребуется увеличить время обучения, что не всегда возможно, особенно при динамическом изменении ситуации во время работы модели, описывающей окружение многоагентной системы.

3) Большой объем занимаемой памяти — количество популяций агентов, реализующих различные решения, в общем случае, только растет. Комбинации, не проявившие жизнеспособность на начальных этапах обучения, могут, тем не менее, вносить полезный генетический опыт для агентов-потомков, но требуют дополнительных объемов памяти и вычислительных ресурсов.

Вывод из исследования моделей, определяющих действия агента

Каждый из рассмотренных популярных подходов имеет свои достоинства и недостатки. Требуется найти и реализовать другой математический метод, который позволит избежать критичных недостатков, присущих рассмотренным подходам. Для эффективного практического применения создаваемых систем необходимо обеспечить:

- наличие целевой функции, учитывающей особенности применения модели;
- недетерминированность поведения агентов;
- быструю оценку ресурсов для решения поставленной задачи с заданным уровнем надежности;
- создание средств обучения (тренажеров) для операторов этих систем.

Вероятностная модель поведения прикладной многоагентной системы

Проблемы коллективного поведения и группового управления привлекают внимание исследователей более полувека. Их значимость значительно возросла в последние годы в связи с актуальностью таких задач, как управление коллективом роботов, управление полетом и выполнением миссии группы беспилотных летательных аппаратов и других мобильных систем. Особенно трудными являются задачи управления группами подвижных объектов, которые должны координировать свое поведение в пространстве и кооперироваться для достижения заданного результата.

За прошедшее время подходы к решению подобных задач значительно изменились, требуя, в частности, новых способов формализации, математических моделей и алгоритмов адаптивного управления поведением прикладных распределённых многоагентных систем. Исследования и разработки в этой области ведутся во многих странах, привлекая большое количество специалистов. Разработана теория динамических интеллектуальных систем, основанных на правилах, исследованы вопросы их устойчивости и управляемости [35]. Однако достаточно развитый математический аппарат, приемлемый для практического управления поведением группы составляющих систему агентов, к настоящему времени в полном объёме пока не создан.

Данная работа представляет собой попытку создать необходимый математический аппарат для частного класса многоагентных систем, область практического применения которых очевидна и не нуждается в комментариях. В ней рассмотрена модель поведения прикладной системы,

представляющей игровое взаимодействие множества агентов и цели [36], которая опирается на описание движения агентов марковскими случайными процессами с дискретными состояниями и непрерывным временем, а также на методы и подходы к решению различных задач, рассмотренные в работах. Поведение агентов является недетерминированным и, поэтому, непредсказуемым с точки зрения цели. Система допускает как согласованное, так и автономное поведение агентов, в зависимости от того, получают или нет агенты информацию о наличии и положении других работоспособных агентов. Поведение агента детализируется алгоритмом, который предусматривает идентификацию параметров вероятностной модели с использованием максимизируемых целевых функций, выражающих групповую и индивидуальные вероятности поражения цели.

Общее описание поведения системы

Агенты L_k $(k = 0, \dots, K-1)$ перемещаются по плоскому игровому полю, на котором находится одна неподвижная цель T, согласно представленным далее правилам, стараясь поразить цель. Для определения их положения вводится разбиение области поверхности, прилегающей к цели, на ячейки, образованные пересечением m концентрических колец и n секторов, причём цель T находится в центре внутреннего кольца (рис. 1). Положение агентов определяется с точностью до ячейки (i, j), индексы которой задаются номером кольца i $(i = 0, \dots, M-1)$ и номером сектора j $(j = 0, \dots, N-1)$. Вероятность пребывания агента L_k в ячейке (i, j) в момент времени t описывается функцией $p_{k,ij}(t)$.

В дискретные моменты времени, разделённые *интервалом дискретизации* Δt, агент L_k может атаковать и, возможно, поразить цель T, а

также быть атакованным и, возможно, поражённым этой целью с определёнными вероятностями, зависящими от его положения. Задаются пороговые вероятности индивидуальной p_t и коллективной p_{tn} атаки цели, пороговая вероятность завершения игры p_{tmax}, наибольшая допустимая вероятность поражения агентов p_B и наибольшая допустимая скорость перемещения агентов по игровому полю v_{max}, а также число агентов B, одновременно атакуемых целью.

В каждый дискретный момент времени все работоспособные агенты располагают информацией о том, в каких ячейках они находятся. В зависимости от игровой ситуации, в указанные моменты времени агенты могут получать или не получать информацию о наличии и положении других работоспособных агентов.

Изменение распределения и состава агентов на игровом поле, происшедшее при переходе от одного дискретного момента времени к другому моменту, следующему за ним по порядку с интервалом Δt, будем называть *тактом игры*.

В зависимости от возможностей агентов получать релевантную информацию, *условием завершения игры* является превышение расчётной вероятностью поражения цели заданного порога, получение информации о поражении цели или поражение целью всех агентов. Первые два исхода игры рассматриваются как *победа агентов*, а третий – как *победа цели*.

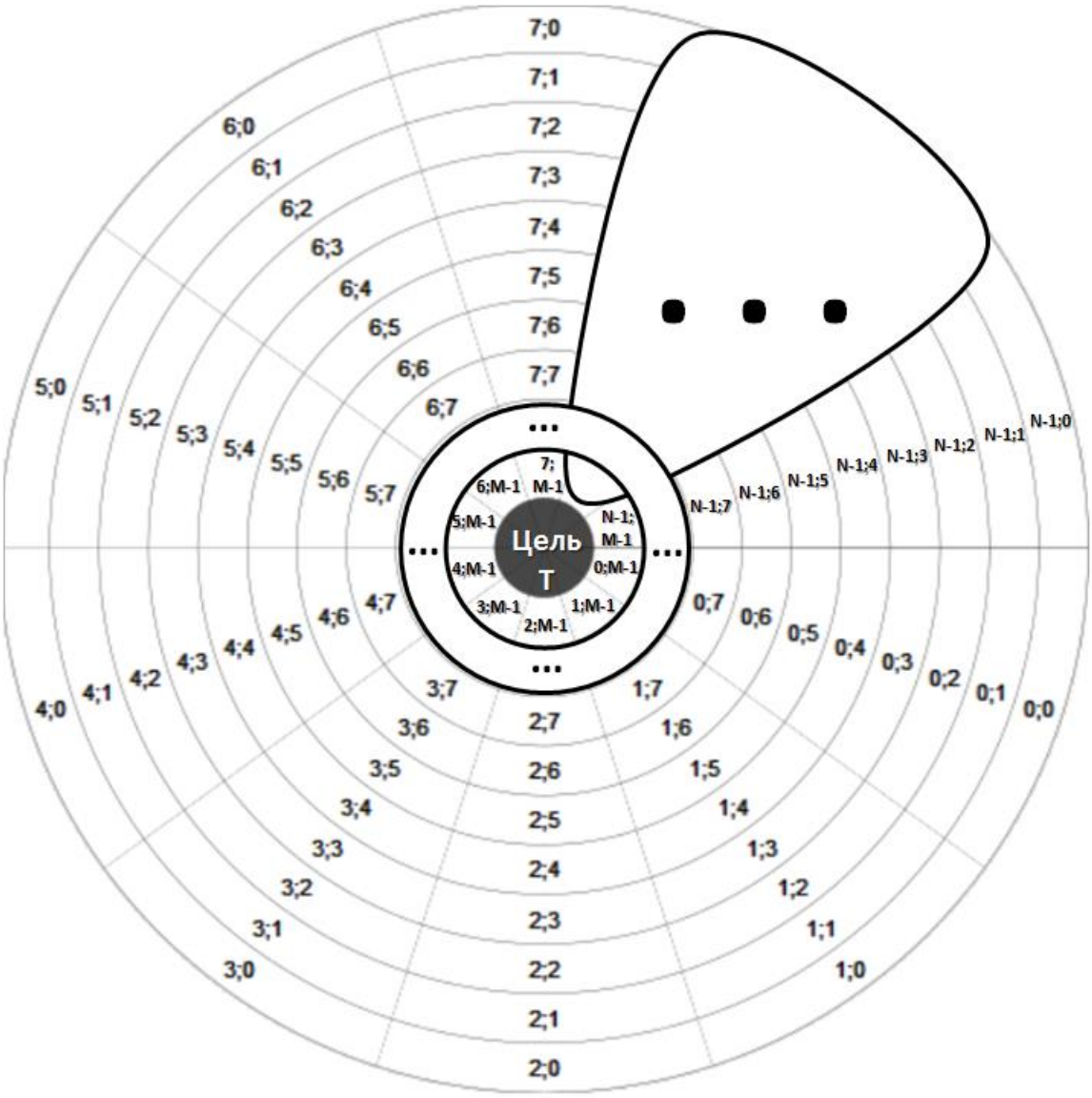

Рис.1. Структура игрового поля.

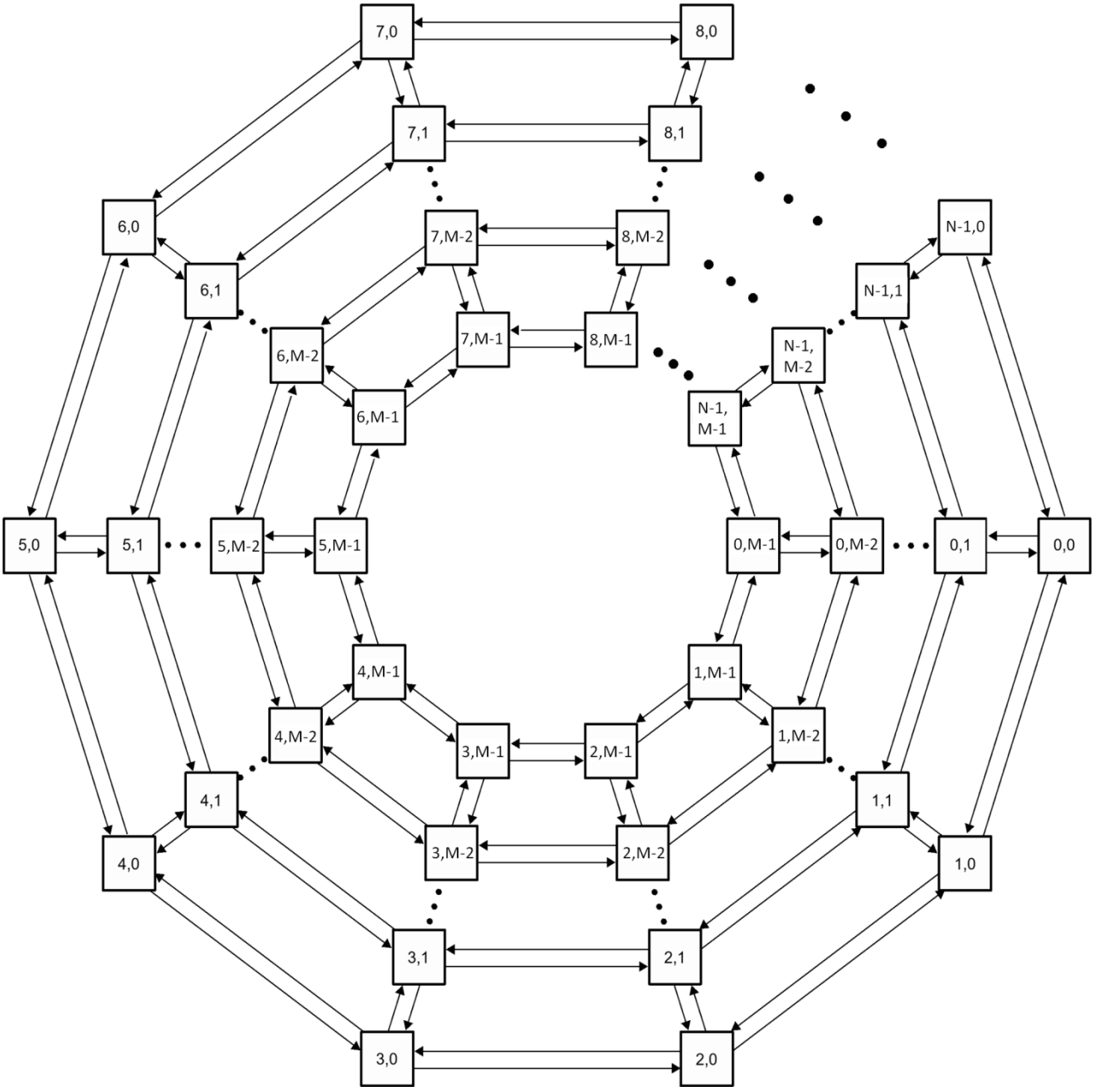

Рис. 2. Структура связей между состояниями марковского случайного процесса, представляющего перемещения агентов по ячейкам игрового поля.

Будем полагать, что перемещение каждого агента по ячейкам рассматриваемой области поверхности описывается *марковским случайным процессом с дискретными состояниями и непрерывным временем*. Пребывание в состоянии марковского процесса соответствует пребыванию в

ячейке указанной области поверхности, имеющей те же самые индексы, а переходы, для которых выполняются свойства пуассоновских потоков событий, возможны только между состояниями, соответствующими *смежным ячейкам*, имеющим общую границу, не совпадающую с точкой. Модель этого процесса может быть представлена ориентированным графом. Обозначив текущий момент времени как t, определим следующие этапы эволюции этой системы.

Этап 1. Задание начального пространственного распределения агентов при $t = 0$ (*этап инициализации*).

Этап 2. Получение агентами информации о состоянии и пространственном расположении друг друга в момент времени t (*первый этап, реализующий взаимодействие агентов*).

Этап 3. Расчёт вероятностного распределения всех работоспособных агентов в пространстве в момент времени $t + \Delta t$, обеспечивающего экстремальное значение заданной целевой функции, зависящей от состояния и текущего пространственного распределения всех агентов, путём решения задачи оптимизации (*этап оптимизации и одновременно второй этап, реализующий взаимодействие агентов*).

Этап 4. Перемещение агентов в позиции, соответствующие их новому распределению в пространстве, вычисленному на этапе 3, в течение интервала времени Δt (*этап перемещения*).

Этап 5. Получение работоспособными (то есть не пораженными целью и способными выполнять действия, определенные моделью) агентами информации о состоянии и пространственном расположении друг друга в

текущий момент времени t (*третий этап, реализующий взаимодействие агентов*).

Этап 6. Согласованная реализация работоспособными агентами попытки поражения цели в случае выполнения заданных условий, наложенных на значения целевой функции (*этап атаки*).

Этап 7. Получение агентами информации о состоянии цели; переход к этапу 2 в случае неудачной попытки поражения цели или завершение игры в случае удачной попытки её поражения или отсутствия работоспособных агентов (*этап контроля*).

Взаимодействие агентов данной системы обеспечивается на этапах 2, 3 и 5. Перемещение каждого агента по ячейкам рассматриваемой области поверхности описывается *марковским случайным процессом с дискретными состояниями и непрерывным временем*, структура которого приведена на рисунке 2. Полагается, что для указанных процессов заданы начальные распределения агентов по ячейкам, а интенсивности переходов между состояниями являются неизвестными (свободными) параметрами моделей.

Число переходов между смежными состояниями X, попадающих в любой временной интервал τ, начинающийся в момент t, распределено согласно *закону Пуассона*:

$$P_{t,\tau}(X = m) = \frac{a(t,\tau)^m}{m!} e^{-a(t,\tau)},$$

где $P_{t,\tau}(X = m)$ – вероятность появления m переходов в течение рассматриваемого интервала, $a(t,\tau)$ – среднее число переходов, попадающих в интервал τ, начинающийся в момент времени t. Далее рассматриваются только стационарные потоки переходов, в которых $a(t,\tau) = \eta\tau$, а $\eta = const$

есть интенсивность стационарного потока. Указанное выше предположение о пуассоновском распределении переходов между смежными состояниями обычно для прикладных задач, поскольку это распределение часто встречается на практике благодаря предельным теоремам для потоков событий.

Поведение каждого агента определяется автономно. Динамика изменения вероятностей пребывания k-го агента в состояниях марковских процессов определяется системой *уравнений Колмогорова* в матричной форме:

$$\frac{d\mathbf{p}_{\boldsymbol{k}}(t)}{dt} = \mathbf{M}_{\boldsymbol{k}}(\boldsymbol{\lambda}_{\boldsymbol{k}})\mathbf{p}_{\boldsymbol{k}}(t)\ ,$$

где $\mathbf{p}_{\boldsymbol{k}}(t)$ представляет вероятности пребывания k-го агента в n состояниях процесса, $\boldsymbol{\lambda}_{\boldsymbol{k}}$ – множество интенсивностей переходов между смежными состояниями для k-го агента, $\mathbf{M}_{\boldsymbol{k}}$ – имеющая порядок n матрица интенсивностей переходов между состояниями для k-го агента. Начальные условия: $p_{k,i_0j_0}(0) = 1, \{p_{k,ij}(0) = 0\}_{i\neq i_0, j\neq j_0}$, где (i_0, j_0) – индексы ячейки, в которой k-й агент находится при $t = 0$.

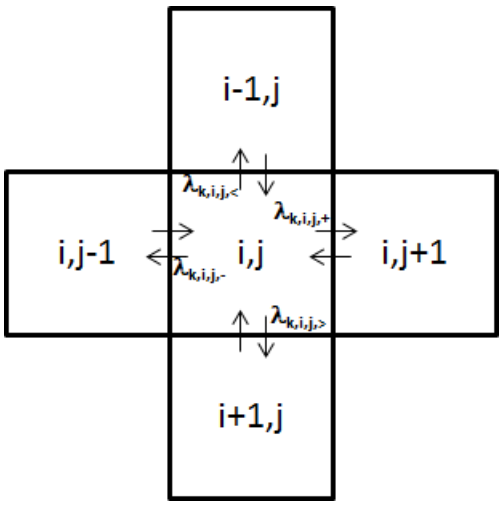

Рис. 3. Обозначения для интенсивностей переходов k-го агента из заданного состояния.

Далее используются следующие обозначения для элементов множества $\boldsymbol{\lambda_k}$ (рис. 3):

- интенсивности переходов k-го агента из состояния (i, j) вдоль i-го кольца состояний по часовой стрелке – $\lambda_{k,i,j,+}$;
- интенсивности переходов k-го агента из состояния (i, j) вдоль i-го кольца состояний против часовой стрелки – $\lambda_{k,i,j,-}$;
- интенсивности переходов k-го агента из состояния (i, j) вдоль j-го сектора состояний по направлению к центру игрового поля – $\lambda_{k,i,j,>}$;
- интенсивности переходов k-го агента из состояния (i, j)вдоль j-го сектора состояний по направлению от центра игрового поля – $\lambda_{k,i,j,<}$.

Расчёт вероятностей $\mathbf{p_k}(t)$ для всех агентов выполняется синхронно в дискретные моменты времени с шагом Δt. Допускается нахождение нескольких агентов в одной и той же ячейке.

Обозначив текущий момент игрового времени как t_*, введём для происходящих событий следующие обозначения:

- A – поражение цели в случае её атаки;
- B_k – поражение k-го агента целью в случае его атаки;
- D_k – атака цели k-м агентом;
- C_k – поражение цели в случае её атаки k-м агентом;

- H_{ijk} – пребывание k-го агента в ячейке (i,j),
- $\widetilde{H}_{ijk}$ – пребывание k-го агента в момент времени $t_* + \Delta t$ в ячейке (i,j), смежной по отношению к ячейке, в которой этот агент находился в момент времени t_*,
- $E_{k,i} = H_{i0k} + \cdots + H_{i,N-1,k}$ – пребывание k-го агента в i-м кольце,
- C_k – переход из ячейки, в которой k-й агент находился в момент времени t_*, в одну из смежных по отношению к ней ячеек.

Вероятность поражения цели в случае её атаки k-м агентом рассчитывается по формуле полной вероятности:

$$p(C_k) = \sum_{i,j} p\left(A\middle|H_{ijk}\right)p(H_{ijk}).$$

Вероятность поражения цели k-м агентом при атаке из ячейки (i,j) в момент времени t определяется «*картой осуществимостей*», представленной функцией f_a:

$$p\left(A\middle|H_{ijk}\right) = f_a(i,j,t).$$

Вероятности $p\left(H_{ijk}\right)$ вычисляются путём решения приведённой выше системы уравнений Колмогорова.

Вероятность поражения целью k-го агента в ячейке (i,j) в момент времени t определяется «*картой уязвимостей*», представленной функцией f_b:

$$p\left(B_k\middle|H_{ijk}\right) = f_b(i,j,t).$$

Карты осуществимостей и уязвимостей пересчитываются на каждом такте игры, отслеживая перемещение цели, что позволяет учитывать её движение по игровому полю. Распределения вероятностей, которые задают

эти карты, при решении многих прикладных задач целесообразно задавать как произведение двух логистических функций, а именно:

$$f_a(i,j,t_*) = c_a\left(\frac{e^{r_{a,d}d_{ij}+q_{a,d}}}{1+e^{r_{a,d}d_{ij}+q_{a,d}}}\right)\left(\frac{e^{r_{a,h}h_{ij}+q_{a,h}}}{1+e^{r_{a,h}h_{ij}+q_{a,h}}}\right),$$

$$f_b(i,j,t_*) = c_b\left(\frac{e^{r_{b,d}d_{ij}+q_{b,d}}}{1+e^{r_{b,d}d_{ij}+q_{b,d}}}\right)\left(\frac{e^{r_{b,h}h_{ij}+q_{b,h}}}{1+e^{r_{b,h}h_{ij}+q_{b,h}}}\right),$$

где $d_{ij}-$ расстояние между центром ячейки (i,j) и целью, $h_{ij}-$ разность высот между центром ячейки (i,j) и целью; параметры c_a, c_b, $r_{a,d}$, $q_{a,d}$, $r_{b,d}$, $q_{b,d}$ идентифицируются методом максимального правдоподобия по имеющимся эмпирическим данным так, чтобы обеспечить наибольшую вероятность наблюдаемых удачных и неудачных попыток поражений цели и агентов в контрольной серии опытов.

Найдём закон распределения времени $\tau_* > 0$, которое затрачивается на переход между состояниями процесса. Вероятность того, что переход не произойдёт, есть $P_\tau(X=0) = e^{-\eta\tau}$. Эта величина равна вероятности того, что $\tau_* > \tau$: $P(\tau_* > \tau) = e^{-\eta\tau}$. Отсюда $P(\tau_* \le \tau) = 1 - P(\tau_* > \tau) = 1 - e^{-\eta\tau}$, где $F(\tau,\eta) = P(\tau_* \le \tau)$ есть функция распределения случайной величины τ_*. Плотность распределения этой величины есть $\eta e^{-\eta\tau}$, а математическое ожидание равно $\int_0^\infty t\eta e^{-\eta\tau}dt = \frac{1}{\eta}$.

Перемещения между смежными ячейками выполняются со скоростью $\vec{v} = (v_\lambda, v_\mu)$, имеющей следующие компоненты:

$$v_\lambda = \Delta l_\lambda/\tau_{\lambda*}, v_\mu = \Delta l_\mu/\tau_{\mu*},$$

где Δl_λ и Δl_μ, соответственно, есть *расстояния между центрами смежных ячеек* в радиальном и трансверсальном направлении, а $\tau_{\lambda*}$ и $\tau_{\mu*}$ –

времена, затрачиваемые на преодоление указанных расстояний. Эти перемещения определяют переходы между состояниями марковского процесса в соответствующих направлениях. В процессе игры времена $\tau_{\lambda*}$ и $\tau_{\mu*}$ генерируются как значения случайных величин, имеющих, в зависимости от выбранного направления, одну из следующих функций распределения: $F(\tau, \lambda_{k,i,j,+})$, $F(\tau, \lambda_{k,i,j,-})$, $F(\tau, \lambda_{k,i,j,>})$ или $F(\tau, \lambda_{k,i,j,<})$.

Вероятность поражения цели при коллективной атаке $p(A)$ определяется по формуле сложения вероятностей. События C_i и C_j полагаются независимыми при $i \neq j$. Допускается расположение нескольких агентов в одной и той же ячейке.

Реализация этапов 2, 5 и 7 эволюции системы обеспечивается имеющимися техническими средствами и лежит вне математической постановки задачи. Задача, решаемая на этапах 1, 3, 4 и 6, формулируется следующим образом.

Даны:

(1) структура связей между состояниями марковского случайного процесса, представляющего перемещения агентов по ячейкам игрового поля;

(2) распределение всех работоспособных агентов по состояниям данного марковского случайного процесса в момент времени t;

(3) функция $f_a(i, j, t)$, определяющая вероятность поражения цели при атаке из ячейки (i, j) в момент времени t;

(4) функция $f_b(i, j, t)$, определяющая вероятность поражения целью агента в ячейке (i, j) в момент времени t;

(5) пороговые вероятности индивидуальной p_t и коллективной p_{tn} атаки цели;

(6) пороговая вероятность завершения игры p_{tmax};

(7) наибольшая допустимая вероятность поражения агентов p_B;

(8) наибольшая допустимая скорость перемещения агентов по игровому полю v_{max};

(9) число агентов *B*, одновременно атакуемых целью.

Найти: распределение всех работоспособных агентов по состояниям заданного марковского случайного процесса в момент времени $t + \Delta t$, обеспечивающее наибольшее значение вероятности поражения цели

- при коллективной атаке в случае наличия информации о положении работоспособных агентов или
- при индивидуальной атаке в случае её отсутствия

при условии

- выполнения всех ограничений, заданных параметрами (5)-(7) условия задачи,
- оценки вероятностей поражения цели и агентов с помощью функций $f_a(i,j,t)$ и $f_b(i,j,t)$и
- недетерминированного перемещения агентов по состояниям марковского процесса.

Для решения данной задачи разработан специальный алгоритм. Следует отметить, что при его выполнении не решается математическая

задача управления в классической постановке, однако численное решение задачи оптимизации является одним из основных компонентов.

Эволюция системы определяется следующим алгоритмом.

- Шаг **1**. Задать начальные условия задачи.
- Шаг **2**. Для текущего расположения агентов в текущий момент времени t_* определить не более B агентов, имеющих наибольшие вероятности поражения целью в соответствии с «картой уязвимостей» $f_b(i,j)$ и случайным образом, соразмерно этим вероятностям, удалить с игрового поля часть указанных агентов; проверить критерии индивидуальной $(p(\mathrm{C}_k) \geq p_t)$ или коллективной $(p(\mathrm{C}_1 + \cdots + \mathrm{C}_n) \geq p_{tn})$ атаки цели агентами; если хотя бы один из них выполнен, то атаковать цель.
- Шаг **3**. Если в момент времени t_* выполнено хотя бы одно из условий завершения игры (исход, приводящий к поражению всех агентов либо цели), завершить игру (перейти к шагу 6).
- Шаг **4**. Выполнить идентификацию значений свободных параметров марковских процессов $\{\boldsymbol{\lambda_k}\}_{k=0,\ldots,K-1}$ при условии $|\vec{v}_k| \leq v_{max}$. Если агенты имеют возможность получать релевантную информацию друг о друге, то максимизируемой *целевой функцией игры*, определяющей результаты этой идентификации, является вычисленная с учётом всех агентов групповая вероятность поражения цели $p(\mathrm{C}_1 + \cdots + \mathrm{C}_n)$ в момент времени $t_* + \Delta t$; в противном случае задача идентификации решается для каждого агента автономно, причём в качестве максимизируемых целевых функций игры используются

индивидуальные вероятности поражения цели $p(\mathrm{C}_k)$ в тот же момент времени $t_* + \Delta t$.

- Шаг **5**. Выбрать одну из ячеек игрового поля, смежных по отношению к ячейке, в которой находится k-й агент в момент времени t_*. Использовать «*метод рулетки*» [37] с вероятностями выбора объектов, пропорциональными прогнозируемым байесовским оценкам

$p\left(\widetilde{H}_{ijk}\,\middle|C_k\right) = \frac{p\left(C_k\middle|\widetilde{H}_{ijk}\right)p(\widetilde{H}_{ijk})}{p(C_k)} = \frac{p(\widetilde{H}_{ijk})}{p(C_k)}$, где вероятности пребывания k-го агента в момент времени $t_* + \Delta t$ в ячейке (i, j), смежной по отношению к ячейке, в которой этот агент находился в момент времени $t_*\, p(\widetilde{H}_{ijk})$, рассчитываются для момента времени $t_* + \Delta t$ как результат выполнения предыдущего шага алгоритма. Переместить в неё этого агента со скоростью $\vec{v}_k$, случайные компоненты которой рассчитываются, используя идентифицированные интенсивности переходов между состояниями, если выполнены следующие ограничения: $|\vec{v}_k| \le v_{max},\ p(B_k) \le p_B$; Если не выполнено ограничение $|\vec{v}_k| \le v_{max}$, то перемещение происходит со скоростью $|\vec{v}_k| = v_{max}$; если не выполнено ограничение $p(B_k) \le p_B$, то перемещение не происходит.

Интервал дискретизации Δt определяется на данном такте игры как наибольшее время, необходимое для перемещения между центрами смежных ячеек: $\Delta t = \max\limits_{k\in\{0,\ldots,K-1\}}(\Delta l_k/|\vec{v}_k|)$, где $\Delta l_k = \sqrt{{\Delta l_{\lambda,k}}^2 + {\Delta l_{\mu,k}}^2}$. Переходы агентов между состояниями синхронизируются по единому (для всех этих объектов) интервалу Δt.

Перейти к следующему по порядку дискретному моменту времени $t_* + \Delta t$, рассматривая его далее как текущее время; перейти к шагу 2.

- Шаг **6**. Завершить игру.

Для решения задачи идентификации, возникающей на шаге 4, разработан численный метод.

Используя рассмотренный подход, можно путём генерации выборки вычислительных экспериментов оценить, сколько агентов необходимо, чтобы поразить цель, а также исследовать зависимость исхода от числа агентов и распределений вероятностей поражения целей и агентов.

Проиллюстрируем работу приведённого выше алгоритма, используя разработанное авторами программное обеспечение.

На рис. 4 и 5 представлены «карта осуществимостей» и «карта уязвимостей», представленные, соответственно, функциями $f_a(i,j)$ и $f_b(i,j)$. В начальный момент времени на игровом поле располагались 10 агентов системы, распределение которых по ячейкам задано на рис. 4 и 5. Вычисленное поведение многоагентной системы, показанное в виде распределений агентов по игровому полю с временным шагом 0,1 *с* на рис. 6-8, определялось следующими параметрами: $p_t = 0{,}8$; $p_{tn} = 0{,}95$; $p_{tmax} = 0{,}95$; $p_B = 0{,}7$. Перемещения агентов по игровому полю и осуществлённые ими атаки цели привели к её поражению на 1 *с* игры.

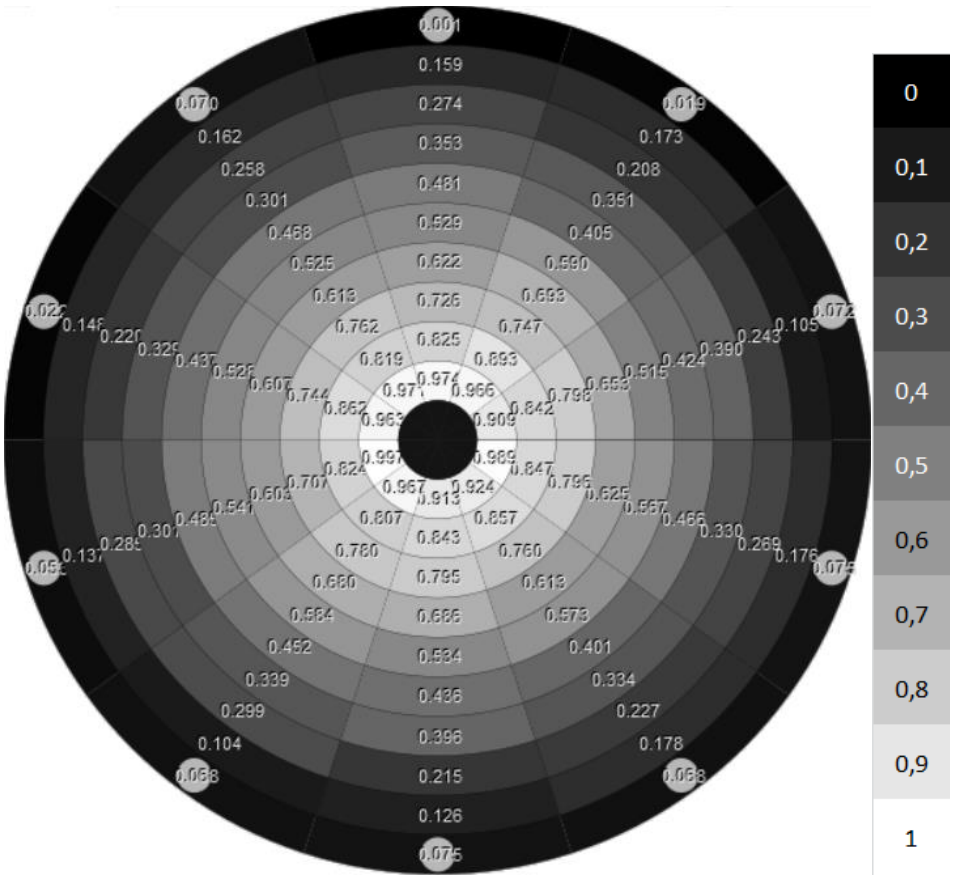

Рис. 4. «Карта осуществимостей» и начальное расположение агентов.

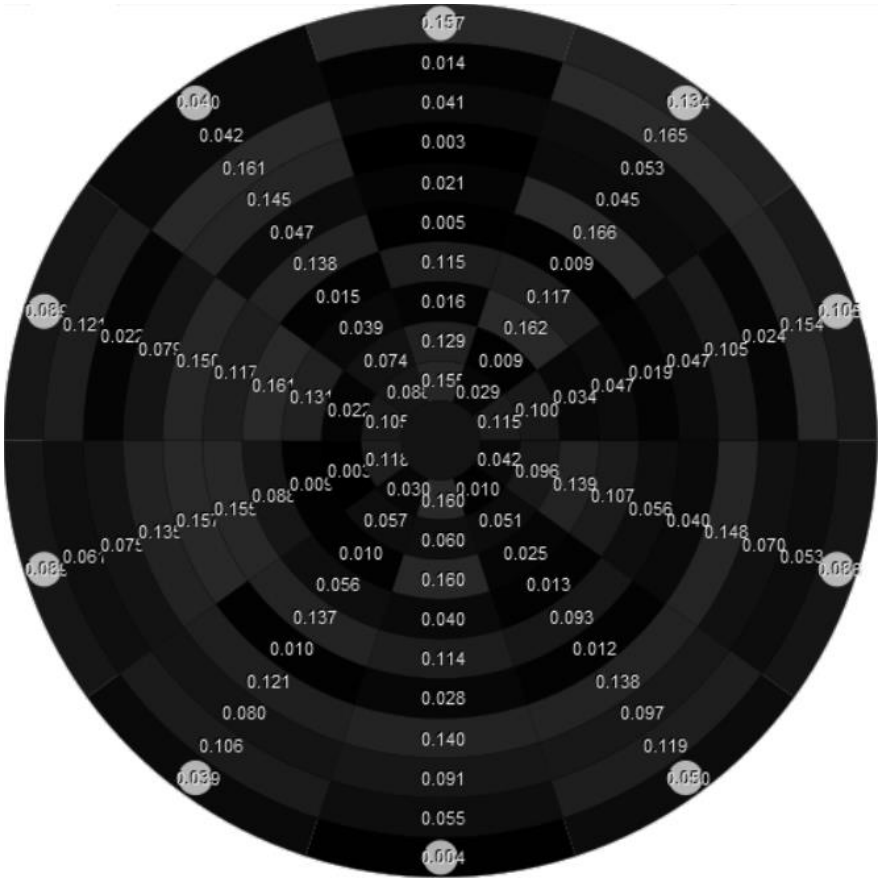

Рис. 5. «Карта уязвимостей» и начальное расположение агентов.

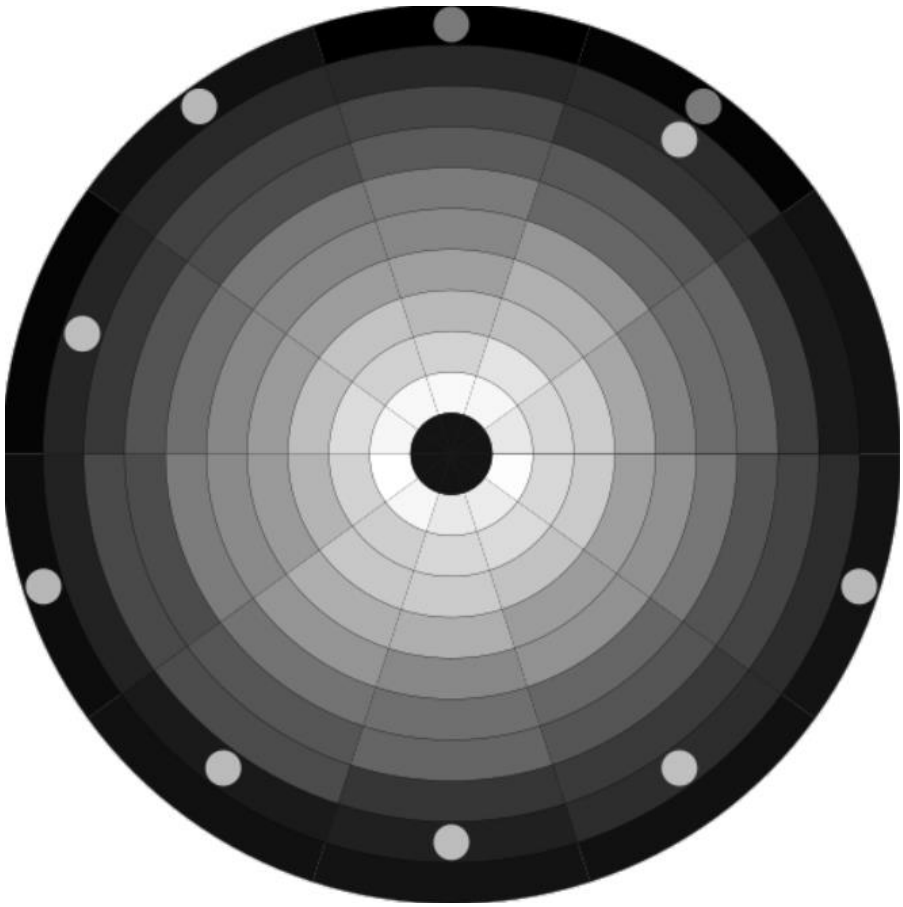

Рис. 6. Первый такт игры: 2 агента на «севере» карты, отмеченные более темным цветом, поражены целью *T*.

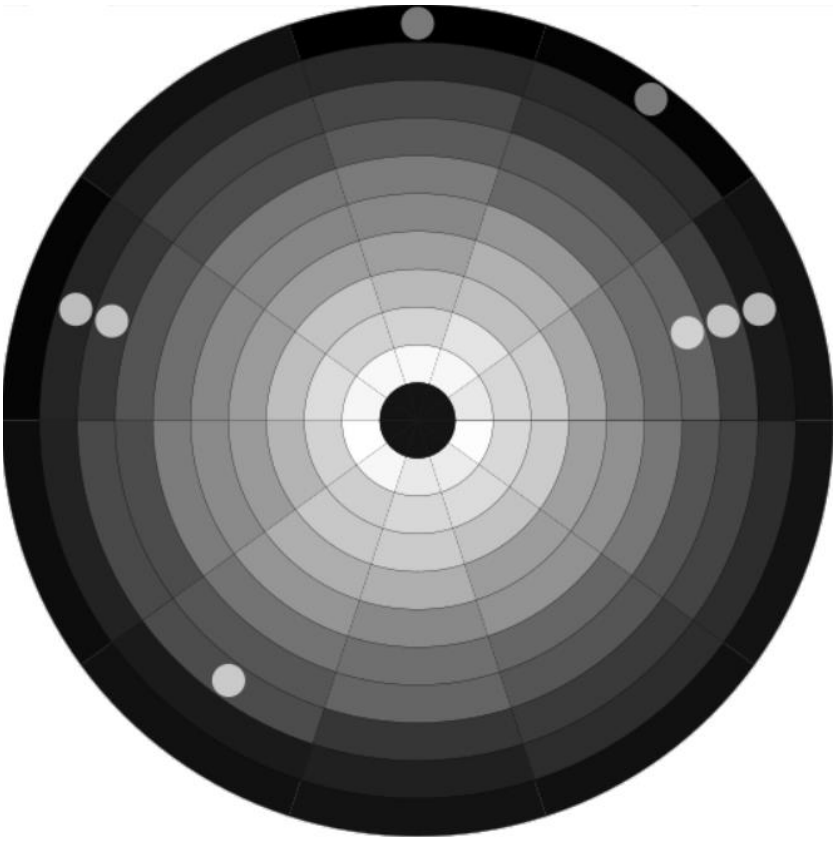

Рис. 7. Пятый такт игры: агенты группируются в ячейках, где выше вероятность попадания в цель *T*.

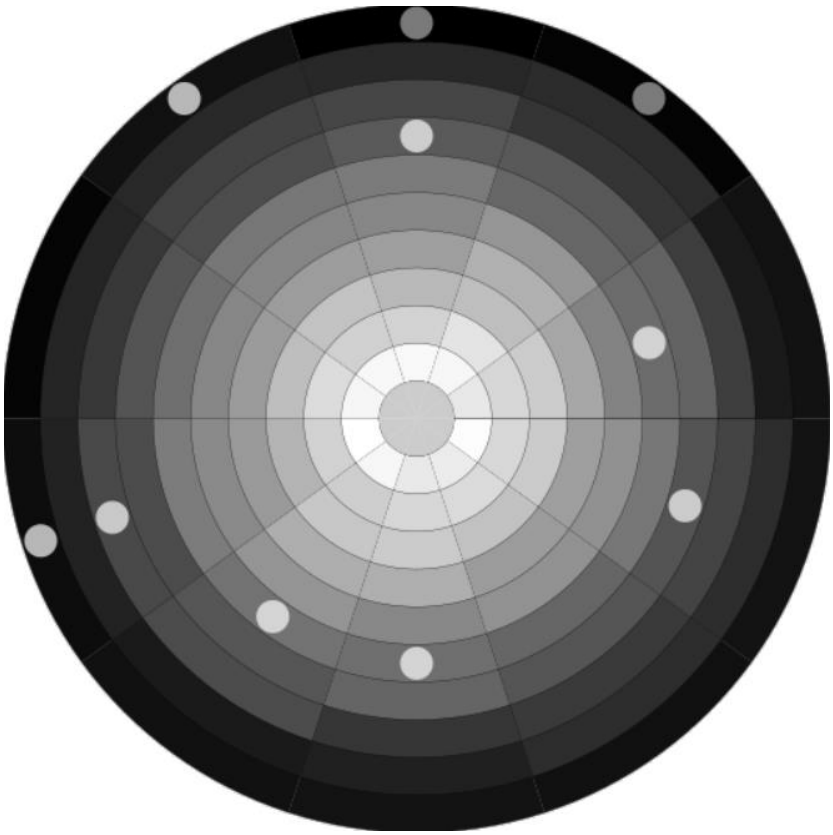

Рис. 8. Десятый такт игры: агенты поражают цель *T*, выполнив коллективную атаку.

Прогноз состояния системы в случае простейшего поведения её агентов

Для прогноза состояния системы в случае простых вариантов развития игры получим аналитические выражения для вероятностей поражения цели в случае её атаки k-м агентом и поражения k-го агента в случае его атаки целью.

Назовём поведение k-го агента *простейшим*, если

– он перемещается к цели, сохраняя постоянную ненулевую интенсивность переходов между смежными кольцами в направлении

цели и нулевую интенсивность переходов между смежными кольцами в противоположном направлении,

– вероятность поражения цели агентом из i-го кольца есть $p\left(A\middle|E_{k,i}\right) = if_{a*}$, где $f_{a*} = const$,

– вероятность поражения целью агента из i-го кольца есть $p\left(B_k\middle|E_{k,i}\right) = if_{b*}$, где $f_{b*} = const$.

Объединим все ячейки каждого кольца игрового поля в укрупнённые состояния и рассмотрим марковский процесс с постоянной интенсивностью переходов, представляющий случайные переходы от одного кольца к другому в случае простейшего поведения (рис. 9). В этом случае интенсивность переходов между смежными кольцами равна сумме интенсивностей переходов между соответствующими парами состояний по секторам (в рассматриваемом случае эти интенсивности не зависят от номеров колец):

$$\lambda_k = \lambda_{k,i} = \sum_{j=0}^{N-1} \lambda_{k,i,j,>}, i = 0, \dots, M-2.$$

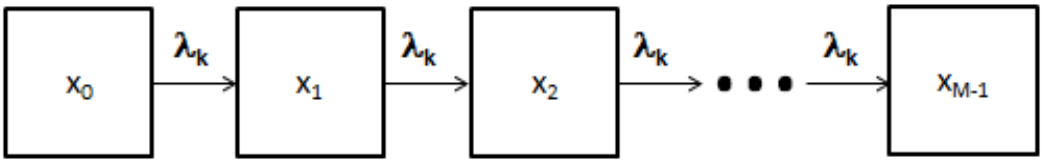

Рис. 9. Марковский процесс с постоянной интенсивностью переходов, представляющий случайные переходы k-го агента от одного кольца к другому в случае простейшего поведения. Состояние x_i $(i = 0, \dots, M-1)$ соответствует пребыванию в i-м кольце игрового поля, λ_k – интенсивность перехода между состояниями

В случае указанного процесса с постоянной интенсивностью λ_k вероятность попадания в i-е состояние в течение временного интервала τ есть вероятность совершения i переходов между состояниями за указанное время:

$$p_\tau\left(E_{k,i}\right) = P_{0,\tau}(X = i) = \frac{(\lambda_k \tau)^i}{i!} e^{-\lambda_k \tau},$$

Множество $\left\{E_{k,i}\right\}_{i=0,\dots,M-1}$ есть полная группа несовместных событий для k-го агента. Вероятность поражения цели этим агентом в течение временного интервала τ определяется по формуле полной вероятности:

$$p_\tau(A) = \sum_{i=0}^{M-1} p\left(A\middle|E_{k,i}\right) p_\tau(E_{k,i}).$$

Последнее выражение может быть преобразовано следующим образом:

$$p(A) = \sum_{i=0}^{M-1} i f_{a*}\, p_\tau\left(E_{k,i}\right) = \sum_{i=0}^{M-1} i f_{a*} \frac{(\lambda_k \tau)^i}{i!} e^{-\lambda_k \tau} = f_{a*} \sum_{i=0}^{M-1} \frac{i(\lambda_k \tau)^i}{i!} e^{-\lambda_k \tau}$$
$$= f_{a*} R(\lambda_k \tau, M),$$

где

$$R(\lambda_k \tau, M) = \sum_{i=0}^{M-1} \frac{i(\lambda_k \tau)^i}{i!} e^{-\lambda_k \tau}.$$

Рассуждая подобным образом, выразим вероятность поражения целью k-го агента в течение временного интервала τ:

$$p_\tau(B_k) = f_{b*} R(\lambda_k \tau, M).$$

Полагая события A и B_k независимыми, получены следующие выражения для событий, имеющих место в течение временного интервала τ:

$$p_\tau(A|\bar{B}_k) = p(A)p(\bar{B}_k) = f_{a*}R(\lambda_k\tau, M)\big(1 - f_{b*}R(\lambda_k\tau, M)\big),$$

$$p_\tau(B_k|\bar{A}) = p(B_k)p(\bar{A}) = f_{b*}R(\lambda_k\tau, M)\big(1 - f_{a*}R(\lambda_k\tau, M)\big),$$

Таким образом, доказаны следующие утверждения.

Утверждение 1. Вероятность поражения цели k-м агентом в течение временного интервала τ в случае его простейшего поведения и сохранения непоражённым есть $f_{a*}R(\lambda_k\tau, M)\big(1 - f_{b*}R(\lambda_k\tau, M)\big)$.

Утверждение 2. Вероятность поражения целью k-го агента в течение временного интервала τ в случае его простейшего поведения и сохранения цели непоражённой есть $f_{b*}R(\lambda_k\tau, M)\big(1 - f_{a*}R(\lambda_k\tau, M)\big)$.

Графики функций $p_\tau(A|\bar{B}_k)$ и $p_\tau(B_k|\bar{A})$ при различных значениях M, $f_{a*} = 0{,}5$ и $f_{b*} = 0{,}2$ представлены на рис. 10 (а,б).

Очевидно, что в общем случае, когда поведение агентов простейшим не является, задача поддаётся решению только с помощью численных методов.

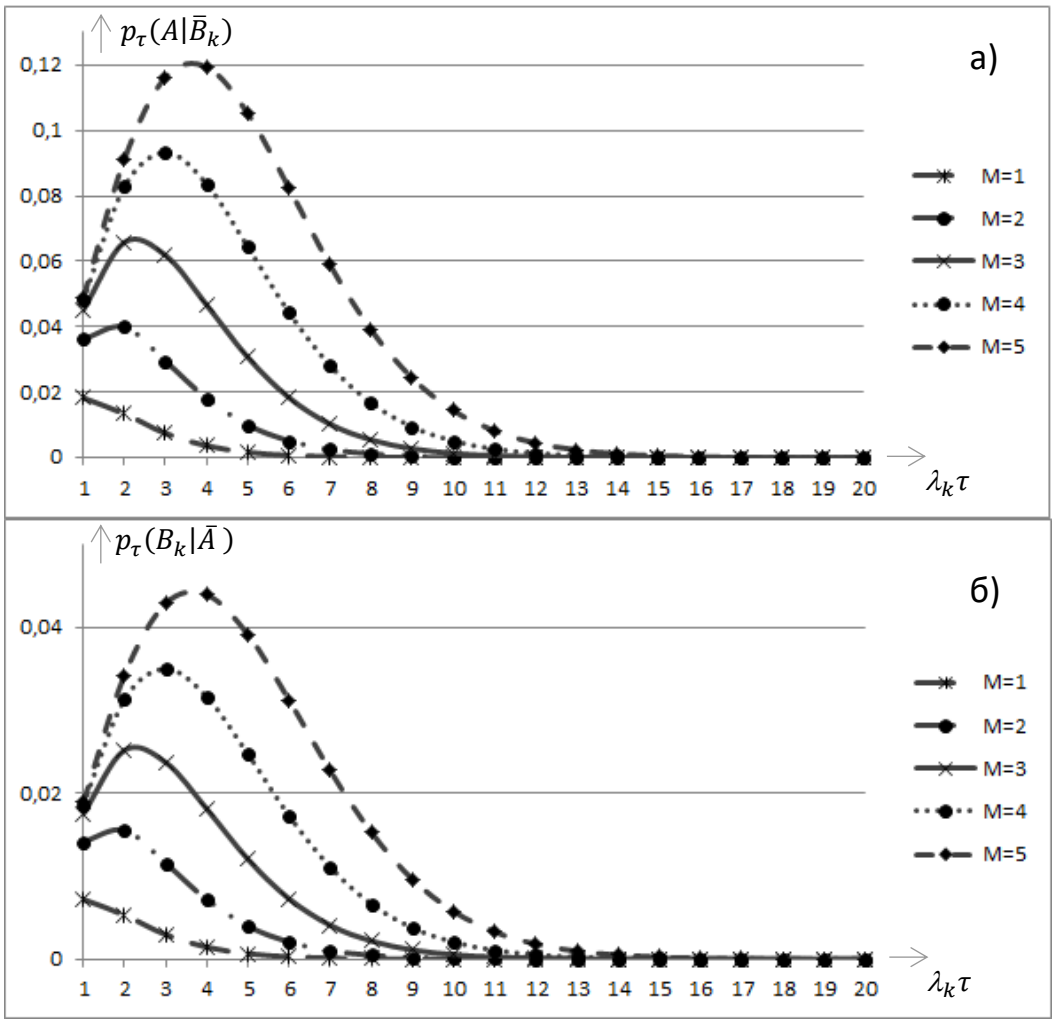

Рис. 10. $p_\tau(A|\bar{B}_k)$ и $p_\tau(B_k|\bar{A})$, представленные, соответственно, на графиках а) и б), как функции $\lambda_k\tau$ при различных значениях M; $f_{a*} = 0{,}5$ и $f_{b*} = 0{,}2$, соответственно.

Представление общих закономерностей поведения многоагентной системы с помощью её макропараметров

Используя рассмотренный подход, можно путём вычислительных экспериментов оценить, сколько агентов необходимо, чтобы поразить цель, а также исследовать зависимость исхода от числа агентов и распределений вероятностей поражения целей и агентов. Однако, учитывая стохастический характер игры, её исход практически остаётся в б*о*льшей или м*е*ньшей степени неопределённым, и однократное моделирование хода игры не может служить основанием для прогноза.

Чтобы обеспечить прогнозирование результатов игры при решении различных практических задач, опираясь на общие характеристики её начальных условий, следует:

- определить удобные для интерпретации и практического контроля макропараметры,
- вычислить для каждого сочетания таких параметров путём имитационного моделирования достаточно представительный ансамбль реализаций хода игры,
- определить статистические характеристики различных её исходов.

Для рассматриваемой игры в качестве таких макропараметров удобно использовать:

- T_{win} – время до выигрыша в тактах,
- L_{lost} – относительное количество поражённых агентов в конце игры,
- N_r – номер ближайшего к цели кольца, в котором располагается агент,

– P_a – количество агентов, располагающихся на игровом поле.

С целью обеспечить независимость прогнозируемого игры результата от конкретного вида игрового поля, первый и четвёртый из указанных параметров рассматриваются в отношении к числу колец. Значения макропараметров N_r и P_a в момент времени $t = i\Delta t$ будем обозначать как N_{ri} и P_{ai}, соответственно.

Проведя достаточное число вычислительных экспериментов и их статистический анализ, результаты такого прогнозирования могут быть представлены в виде зависимостей $T_{win,\gamma} = f_T\left(N_{r0}, P_{a0}\right)$ и $L_{lost,\gamma} = f_L\left(N_{r0}, P_{a0}\right)$, где $T_{win,\gamma}$ и $L_{lost,\gamma}$ – γ-квантили эмпирических распределений величин T_{win} и L_{lost}, N_{r0} и P_{a0} – значения макропараметров в начальный момент времени ($t = 0$), которые графически удобно изображать на плоскости в виде поверхностей уровня.

Знание зависимостей $T_{win,\gamma} = f_T\left(N_{r0}, P_{a0}\right)$ и $L_{lost,\gamma} = f_L\left(N_{r0}, P_{a0}\right)$ позволяет решить *обратную задачу*.

Обратная задача. Найти начальные значения макропараметров N_{r0} и P_{a0}, обеспечивающие выполнение неравенств $T_{win} \le T_*$ и $L_{lost} \le L_*$ с вероятностью $p \ge p_*$.

Для решения этой задачи следует использовать имеющиеся представления зависимостей $T_{win,p^*} = f_T\left(N_{r0}, P_{a0}\right)$ и $L_{lost,p^*} = f_L\left(N_{r0}, P_{a0}\right)$, с помощью которых в области значений макропараметров N_r и P_a

определяется пересечение множеств, заданных неравенствами $T_{win} \le T_*$ и $L_{lost} \le L_*$, что и является искомым результатом.

В терминах указанных макропараметров, вероятностная динамика игры описывается *марковским случайным процессом с дискретными состояниями и дискретным временем*. Каждое состояние этого процесса соответствует сочетанию диапазонов значений макропараметров N_r и P_a (q диапазонов для N_r и l диапазонов для P_a). Интерпретация этих макропараметров делает допустимыми переходы только между смежными состояниями, как в общем виде показано на рис. 3. Число используемых диапазонов определяет точность прогнозирования.

Следует заметить, что, как правило, не все вероятности переходов между состояниями, представленные на рис. 3, являются ненулевыми. В частности, поскольку агенты не восстанавливаются после поражения, переходы в сторону увеличения их численности имеют нулевую вероятность.

Распределения вероятностей пребывания в состояниях в смежные моменты времени $i\Delta t$ и $(i+1)\Delta t$ связаны матричным уравнением:

$$\mathbf{p}_{L,i+1} = \mathbf{L}\mathbf{p}_{L,i},$$

где $\mathbf{p}_{L,i}$ и $\mathbf{p}_{L,i+1}$ представляют вероятности пребывания указанного марковского процесса с дискретными состояниями и дискретным временем в определённых выше lq состояниях в смежные моменты времени $i\Delta t$ и $(i+1)\Delta t$, соответственно; $\mathbf{L} = \left[l_{ij} \right]$ – матрица вероятностей переходов между этими состояниями, имеющая порядок lq. В начальный момент времени компонент $\mathbf{p}_{L,0}$, соответствующий исходному сочетанию диапазонов

значений макропараметров N_r и P_a, равен единице, остальные компоненты этого вектора равны нулю.

Используя указанное матричное уравнение, можно решить *прямую задачу*.

Прямая задача. Задано распределение вероятностей $\mathbf{p}_{L,0}$ пребывания в состояниях марковского процесса, представленного на рис. 3, в начальный момент времени. Найти распределение вероятностей $\mathbf{p}_{L,k}$ в состояниях данного процесса в момент времени $k\Delta t$, где k – натуральное число.

Искомое распределение тривиально вычисляется с помощью матричной формулы:

$$\mathbf{p}_{L,k} = \mathbf{L}^k \mathbf{p}_{L,0} .$$

Моделирование поведения марковских процессов, представленных на рис. 2, позволяет вычислять ансамбли их реализаций, используя которые можно идентифицировать процесс, представленный на рис. 11, вычислив выборочные оценки вероятностей переходов l_{ij} между состояниями рассмотренного процесса с дискретным временем, а именно:

$$l_{ij} = \frac{F_{ij}}{\sum_{k=1}^{lq} F_{kj}},$$

где F_{ij} – наблюдаемая частота переходов из состояния j в состояние i.

Определённый выше процесс с дискретным временем позволяет исследовать общие закономерности динамики игры без привязки к конкретным распределениям агентов по игровому полю и прогнозировать, как изменяются со временем вероятностные распределения значений макропараметров. Переход к описанию хода игры в терминах

макропараметров приводит к определённой потере информации о деталях игры, однако позволяет перейти от воспроизведения конкретных её реализаций к исследованию общих закономерностей, представляющих практический интерес.

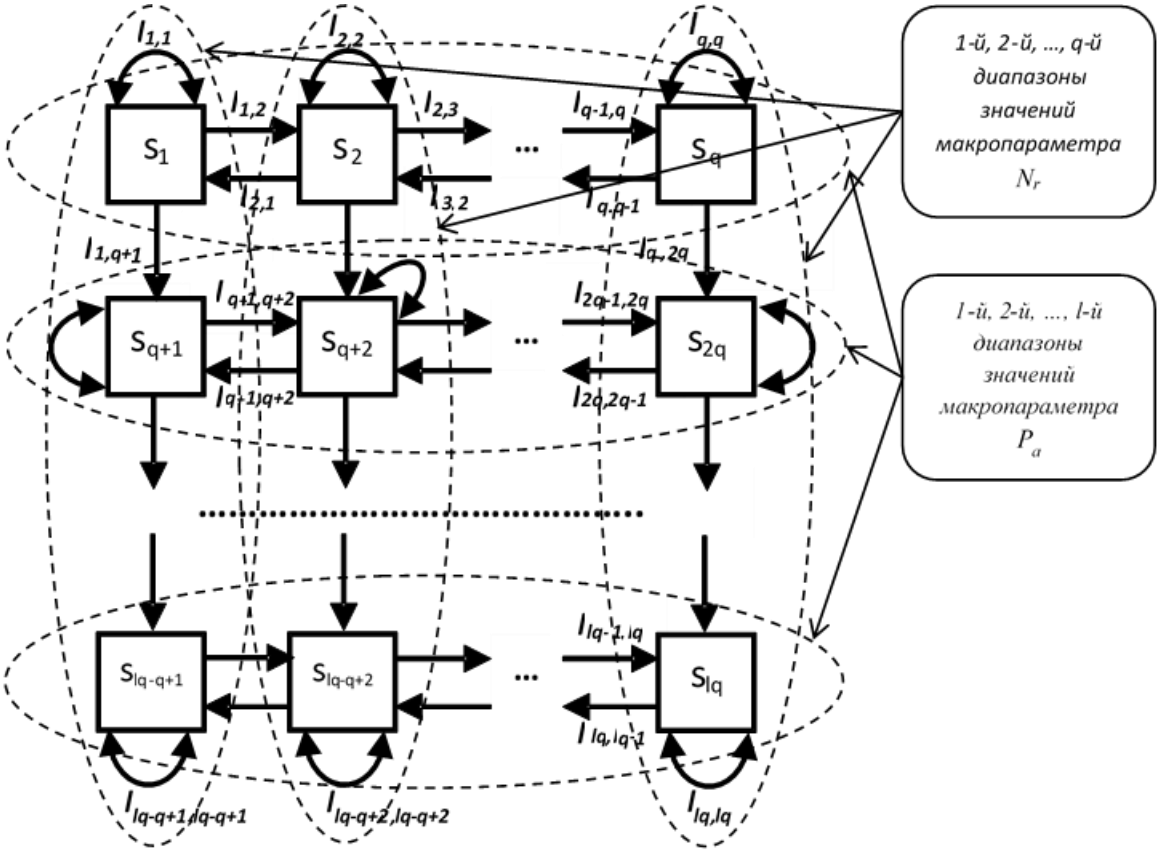

Рис. 11. Марковский случайный процесс с дискретными состояниями и дискретным временем, представляющий вероятностную динамику игры в терминах макропараметров N_r и P_a.

Рассмотрим, в качестве примера, многоагентную систему с параметрами M=N=8 и картами осуществимостей и уязвимостей, приведёнными на рис. 12 и 13, и соответствующий ей марковский процесс с параметрами $q = l = 8$, представляющий вероятностную динамику игры в терминах макропараметров N_r и P_a.

Найдём начальные значения макропараметров N_{r0} и P_{a0}, которые обеспечивают выполнение неравенств $T_{win} \le 1$ (что соответствует не более чем 8 шагам игры) и $L_{lost} \le 0.5$ (что задаёт не более чем 50% потерь) с вероятностью $p \ge 0.95$ (обратная задача).

Зависимости $T_{win,0.95} = f_T\left(N_{r0}, P_{a0}\right)$ и $L_{lost,0.95} = f_L\left(N_{r0}, P_{a0}\right)$, вычисленные по результатам 320 000 вычислительных экспериментов и их статистического анализа, приведены в виде диаграмм на рис. 14 и 15. Пересечение указанных на этих рисунках множеств значений макропараметров N_{r0} и P_{a0}, обеспечивающих, соответственно, выполнение неравенств $T_{win} \le 1$ и $L_{lost} \le 0.5$ с вероятностью $p \ge 0.95$, которое является решением обратной задачи, представлено на рис. 16.

Вычисление выборочных оценок вероятностей переходов между состояниями процесса, представляющего динамику игры в терминах макропараметров, даёт оценку матрицы вероятностей переходов $\mathbf{L}$, позволяя решить прямую задачу, которая сводится к вычислению распределений вероятностей пребывания в состояниях в заданные моменты времени. Примеры таких распределений приведены на рис. 17-18.

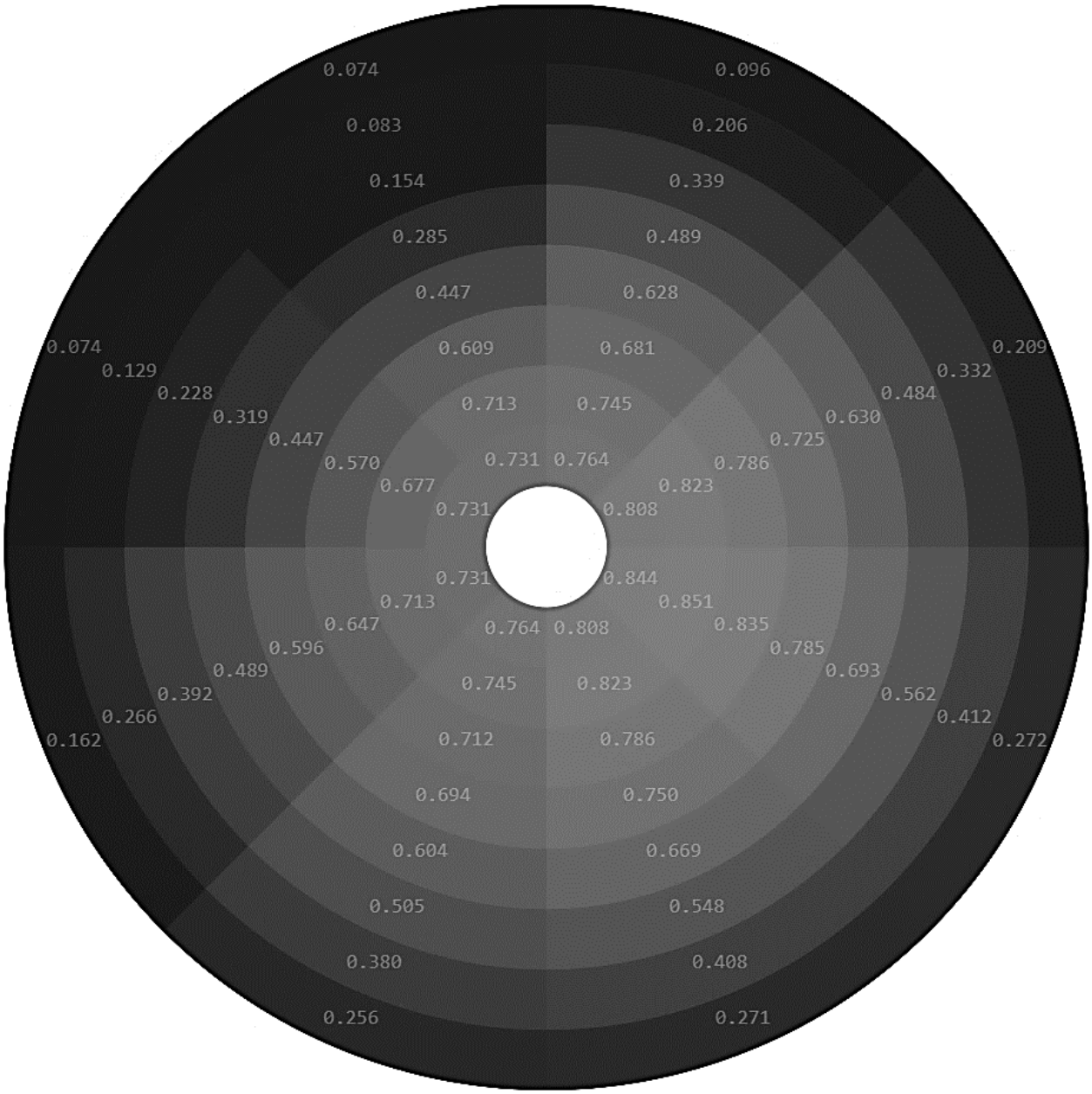

Рис. 12. Карта осуществимостей для многоагентной системы с параметрами M=N=8.

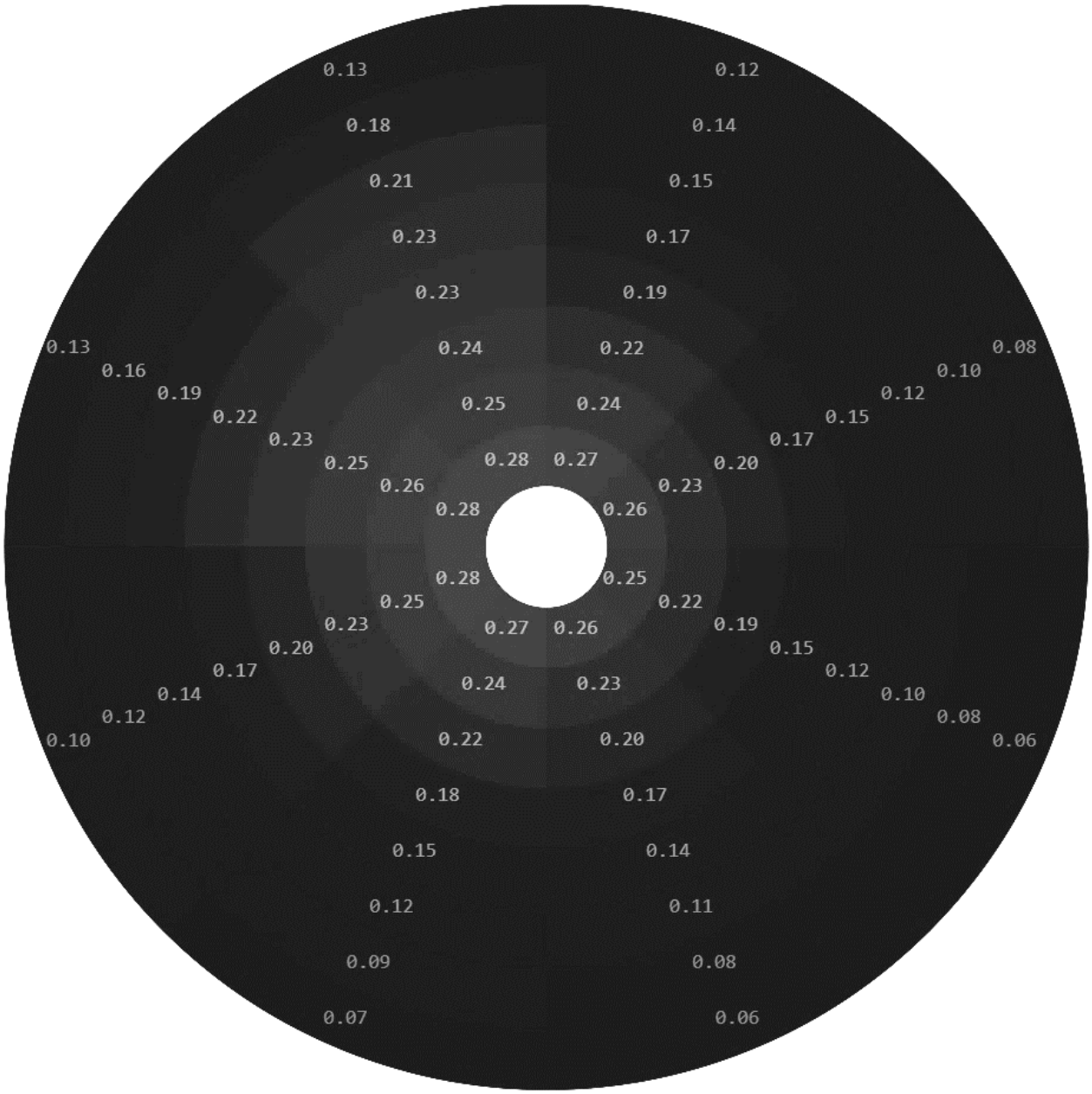

Рис. 13. Карта уязвимостей для многоагентной системы с параметрами M=N=8.

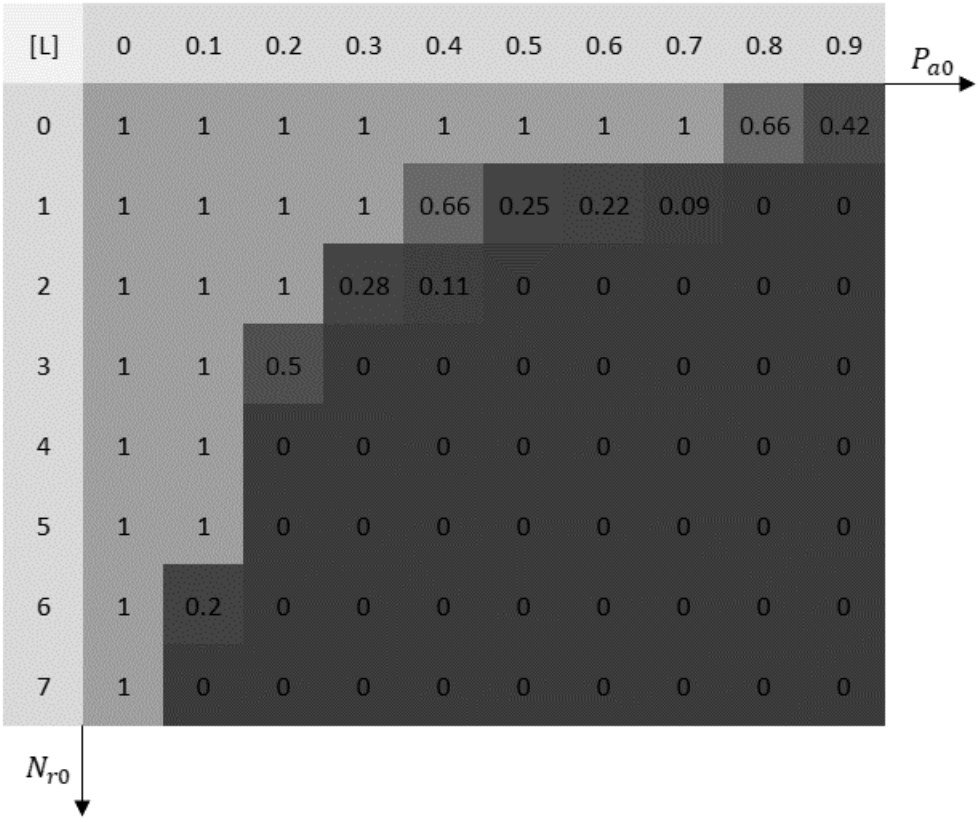

Рис. 14. Диаграмма, представляющая зависимость $L_{lost,0.95} = f_L\left(N_{r0}, P_{a0}\right)$.

Параметр P_{a0} задан в отношении к числу колец.

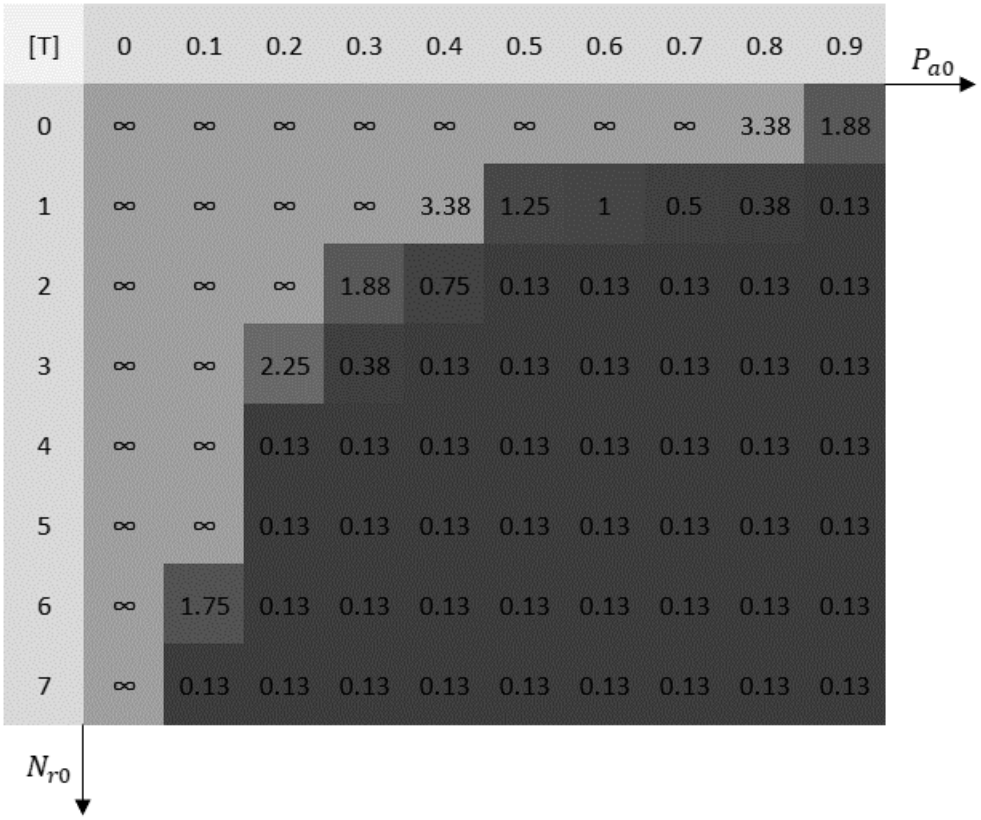

Рис. 15. Диаграмма, представляющая зависимость $T_{win,0.95} = f_T\left(N_{r0}, P_{a0}\right)$.

Параметр P_{a0} и значения функции $T_{win,0.95}$ заданы в отношении к числу колец.

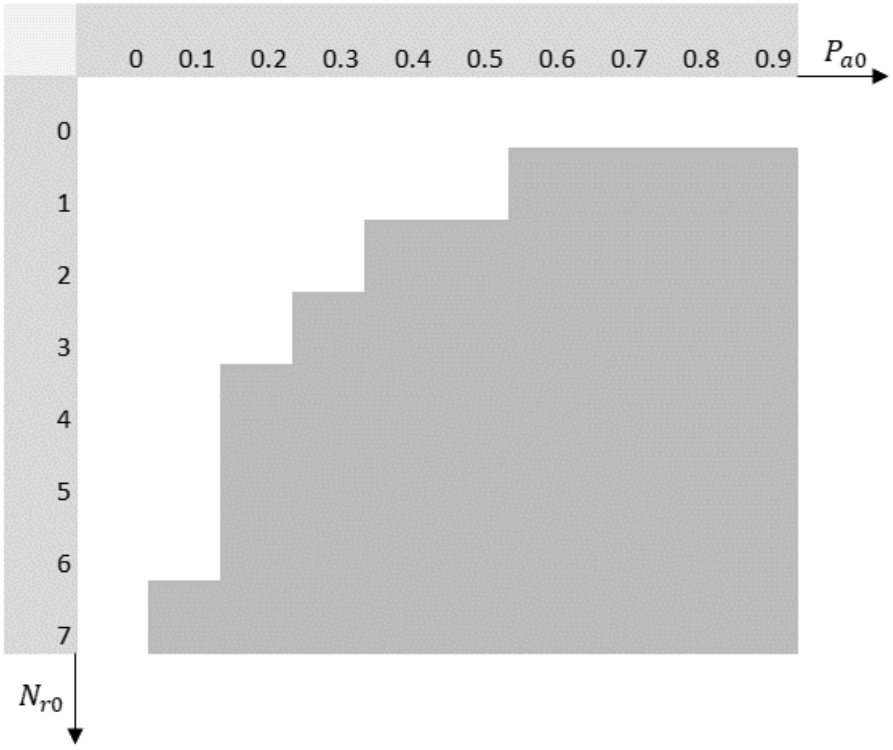

Рис. 16. Пересечение указанных на рис. 14 и 15 множеств значений макропараметров N_{r0} и P_{a0}, обеспечивающих, соответственно, выполнение неравенств $T_{win} \le 1$ и $L_{lost} \le 0.5$ с вероятностью $p \ge 0.95$ (решение обратной задачи).

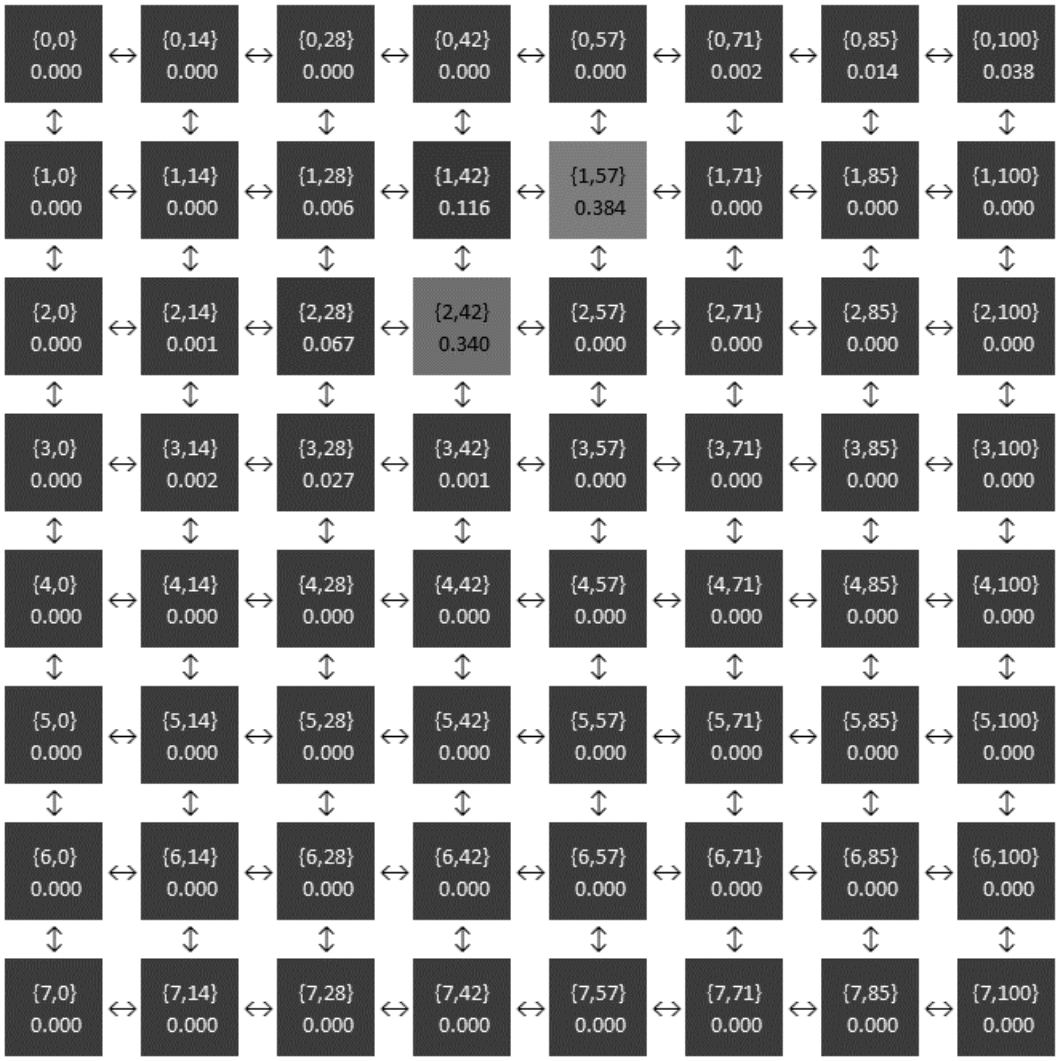

Рис. 17. Оценка распределения вероятностей пребывания в состояниях марковского процесса, представленного на рис. 11, в момент времени $t = 3\Delta t$. Каждое состояние идентифицируется парой индексов, задающих сочетание диапазонов значений макропараметров N_r и P_a (решение прямой задачи).

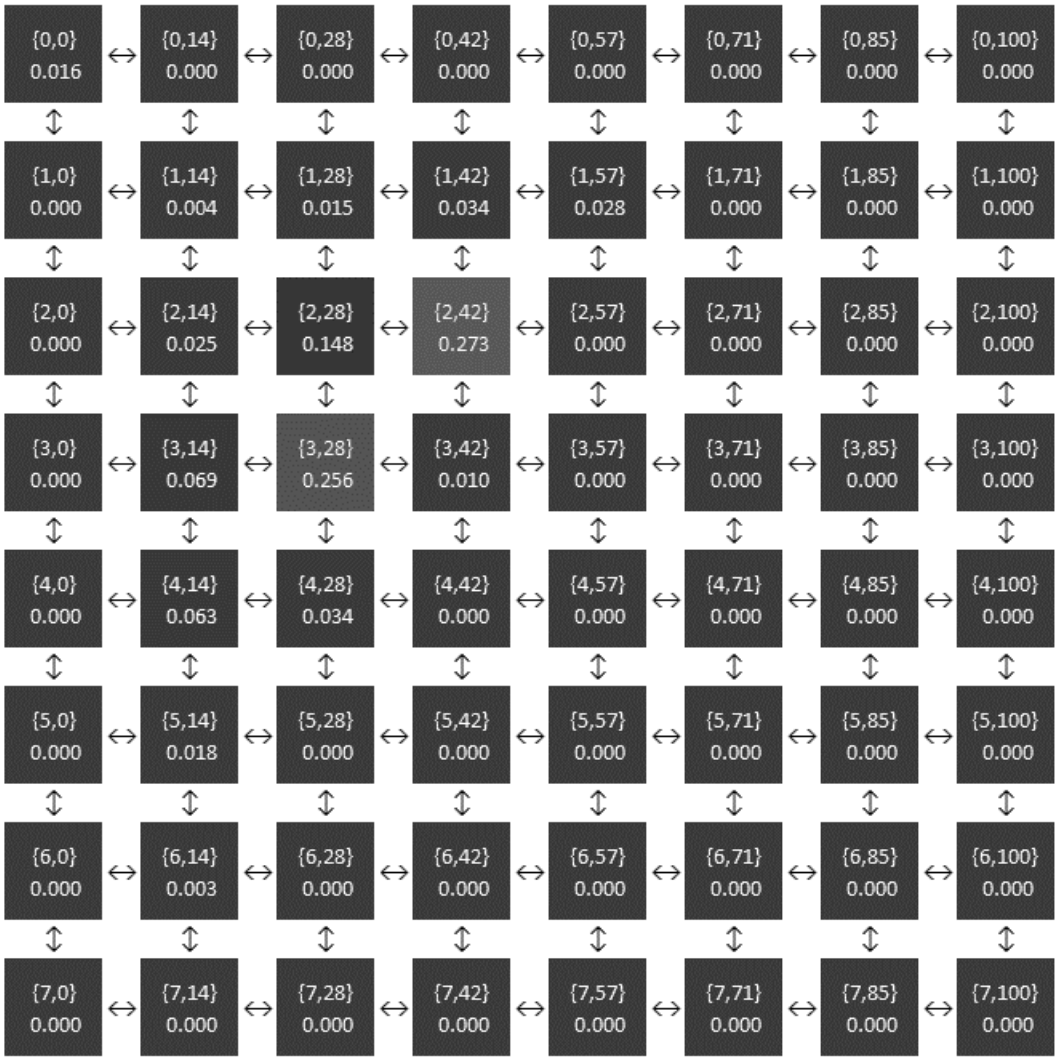

Рис. 18. Оценка распределения вероятностей пребывания в состояниях марковского процесса, представленного на рис. 11, в момент времени $t = 9\Delta t$ (решение прямой задачи).

Программная реализация вероятностной модели поведения прикладной многоагентной системы

Программная версия базовой модели, описывающей многоагентную систему, представляет собой реализацию пяти основных компонентов, связанных между собой: математического аппарата, общего алгоритма действий участников моделируемого процесса (в данном случае, речь идет об агентах и цели), редактора входных данных и параметров модели, проигрывателя модели и подсистемы ввода-вывода для хранения данных.

Перейдем непосредственно к вопросам, связанным с разработкой приложения, обслуживающего данную модель.

Общее описание задействованных в программной реализации модели структур

При программной реализации марковский процесс представляется последовательностью связанных между собой блоков с одним входом (X) и выходом (Y), а также значением вероятности, характеризующим сам блок (P). Вход и выход подсоединяются к следующему блоку в последовательности; последний блок, в случае нециклического соединения блоков, имеет значения входа и выхода равными 0.

Для описания процесса перехода агента из клетки в клетку требуется задать две группы таких соединений, описывающих марковские процессы; одна соответствует переходу по секторам и является циклической, тогда как вторая описывает переход по кольцам и является нециклической. Произведение вероятностей между соответствующими блоками в обоих группах представляет собой вероятность перехода в определенный сектор

заданного кольца, в соответствии с координатами соответствующих блоков в группах соединений.

Нумерация колец и секторов начинается с 0. Кольца нумеруются от самого удаленного от центра кольца, сектора - по часовой стрелке от сектора, наиболее близкого к нулевому градусу.

Обе группы соединений являются частью математического аппарата, позволяющего составлять и решать уравнения Колмогорова для заданных марковских процессов, а также применять расширенный метод Эйлера к ним. Структура каждой такой группы состоит из двоичного флага, определяющего цикличность, а также из массива вещественных чисел, задающего значения X,Y и P для всех N блоков, входящих в состав соответствующей группы.

Совокупность групп является частью структуры, задающей свойства агента в рамках моделируемой игровой ситуации. Также общая структура агента определяется координатами его местоположения (парой целочисленных переменных PX и PY), заданной скоростью и двоичным флагом выхода из строя. Дополнительно вводится двоичный флаг, определяющий первый ход агента, чтобы предотвратить избыточный перерасчет значений вероятностей.

Игровое поле, в котором существуют агенты, задается тремя полями: структурой, представляющей собой строку для имитации помех связи между агентами; словарем, хранящим количество агентов в заданных координатах игрового поля; и массивом вещественных чисел, описывающим карты A (осуществимостей) и B (уязвимостей) для каждого сектора каждого круга игрового поля.

Принципы выбора языков для разработки программной реализации описываемой модели

Для реализации проекта было решено использовать такие технологии, которые бы в явном виде поддерживали следующие принципы: поддерживаемость, простоту, современность, высокую производительность, универсальность, функциональную полноту, кроссплатформенность, независимость, компилируемость и естественную удобочитаемость. Необходимо определиться с языком программирования, на котором будет производиться разработка.

Поддерживаемость языка здесь понимается как наличие сообщества, которое заинтересовано в эксплуатации возможностей языка и гарантирует его развитие в ближайшем времени.

Простота - свойство языка, позволяющее легко овладеть его основами без риска использовать предоставляемые им возможности некорректно.

Универсальность - возможность успешно применять программы, написанные на данном языке, в различных сферах деятельности.

Под функциональной полнотой понимается наличие постоянно пополняемой стандартной библиотеки (либо наличие поддерживаемого сообществом языка хранилища библиотек), функционал которой позволяет языку сохранить актуальность, а программистам - проводить разносторонние исследования в различных областях и сферах деятельности; это условие необходимо, чтобы иметь под рукой мировые общепризнанные решения в случае возможного в дальнейшем расширения поставленной задачи.

Свойство кроссплатформенности необходимо учесть ввиду очевидных намечающихся тенденций в развитии программного обеспечения,

подразумевающих пересечение различных операционных систем и архитектур в своеобразном универсальном вычислительном пространстве. Среди таких тенденций можно отметить платформу WSL, обеспечивающую поддержку приложений Linux в Windows 10, рост популярности платформы .NET Core, а также современные кросс-платформенные облачные решения Microsoft Azure.

Независимость требует отсутствия какой-либо функциональной прослойки, кроме аппаратного обеспечения и установленной целевой операционной системы, для успешного выполнения программ, написанных на данном языке с применением стандартных для этого языка средств. Например, Java по умолчанию требует наличия JVM (“Java Virtual Machine”, “Виртуальная машина Java”) для выполнения программ, тогда как C позволяет создавать программы, не требующие подобных посредников.

Компилируемость - наличие одного или нескольких поддерживаемых разработчиками языка либо его сообществом компиляторов (в противоположность интерпретаторам) для успешного преобразования программы на данном языке к “родному” (англ. “native”) для целевой платформы машинному коду. Предпочтение компиляции обусловлено более высокой производительностью конечного приложения в целом, а также значительно большей степенью защищенности исходного кода от действий потенциальных злоумышленников.

Естественная удобочитаемость - такое построение синтаксиса языка, при котором крайне неудобно или невозможно писать трудные для сопровождения программы (например, наличие отступов в языке Python как обязательного синтаксического элемента).

Современность и высокая производительность в развернутом пояснении не нуждаются.

Комбинацию перечисленных свойств языка программирования, востребованных для разработки программного обеспечения, обеспечивающего работу представляемой системы, можно свести к пяти тезисным требованиям:

— Необходим дружественный к начинающим программистам универсальный язык, на котором можно легко и быстро писать прототипы приложений

— Язык должен обеспечить гибкость построения кода, чтобы его было удобно масштабировать и отлаживать

— Язык должен поддерживать кроссплатформенность по умолчанию для компиляции конечного приложения под разные программно-аппаратные архитектуры, включая разрядность процессора.

— Язык должен иметь поддержку современных математических библиотек и средств для простоты реализации очевидных математических абстракций

— Язык должен обладать современными средствами управления ресурсами (сборщик мусора, аффинная система типов и т.п.), сетевого (средства развертывания серверов) и многопоточного программирования

— Язык должен обеспечивать высокую производительность и позволять приложению работать без зависимостей и потребности в установке

каких-либо сторонних (внешних) дополнительных фреймворков и сред (.NET, JVM и т.п.)

Краткий обзор выбранных для разработки языков

Были рассмотрены такие языки, как C, C++, Java, Object Pascal (Delphi), Free Pascal, CoffeeScript, группа языков платформы .NET (VB, C#, F#) и различные диалекты языка Basic (включая VBA и VBScript), Haskell, Python, Golang, Assembler, Ruby, Io, Prolog, Scala, Erlang, Clojure и другие.

Подробное описание процесса отбора языков выходит за рамки данного материала [38-86]. Краткая таблица соответствия наиболее подходящих для реализации языков представлена на рисунке 19. В результате отбора было решено осуществить программную реализацию модели на языках Python и Golang. Промежуточная коммуникация осуществляется с помощью языка C++.

	Императивные						Декларативные		
	Высокоуровневые					Низкоур.	Функциональные		Логическ.
	C/C++	C#/Java	**Python**	**Golang**	Rust	Assembler	Haskell	Kotlin	Prolog
Дружественность	-	+	+	+	-	-	-	-	-
Быстрое прототипирование	-	-	+	-	-	-	+	+	+
Гибкий, масштабируемый код	-	-	+	+	+	-	+	+	+
Удобство отладки	+	+	+	+	+	-	-	-	-
Кроссплатформенность	+	+	+	+	+	+	+	+	+
Поддержка математических абстракций	-	-	+	-	-	-	+	+	-
Средства управления ресурсами	-	+	+	+	+	-	+	+	+
Средства развертывания серверов	-	+	+	+	-	-	+	+	+
Эффективная поддержка многопоточности	-	+	+	+	+	-	+	+	-
Высокая производительность	+	-	-	+	+	+	-	-	-
Отсутствие зависимостей	+	-	+	+	+	+	+	+	+

Рис. 19. Таблица соответствия отбираемых языков программирования представленным критериям (знак «+» означает соответствие, знак «-» – несоответствие заданному критерию).

Далее будут кратко описаны основные свойства отобранных языков.

C++

C++ - компилируемый язык программирования общего назначения, появившийся как естественное развитие языка C. Занимает лидирующие позиции среди прочих языков программирования: на момент написания главы только Java опережает семейство языков C/C++ по рейтингу TIOBE в связи с доминированием операционной системы Android на рынке мобильных приложений. C++ по-прежнему поддерживается и продвигается на уровне стандарта ISO, включая регулярные обновления стандарта.

В книге “The Design and Evolution of C++” (“Дизайн и эволюция C++”) описаны основные принципы, вошедшие в основу языка. Среди них есть как удовлетворяющие критериям отбора, указанным выше, так и противоречащие. Стремление получить универсальный язык с эффективностью и переносимостью языка C позволило наделить C++ такими качествами, как универсальность, кроссплатформенность, компилируемость и независимость. Решение предоставить разработчику свободу выбора, даже если это позволит ему воспользоваться такой свободой заведомо неправильно, становится предпосылкой к общей сложности языка в ущерб простоте и очевидности. Описанная ранее популярность и твердая поддержка языка обусловлена, не в последнюю очередь, высоким уровнем производительности и гарантирует появление все новых современных библиотек, позволяющих функционально реализовывать разработки в разнообразных сферах. Тем не менее, язык нельзя по-настоящему назвать современным из-за стремления сохранить совместимость с языком C, в том числе, с его устаревшими конструкциями. Синтаксис не только не предполагает удобочитаемость кода, но даже частично поощряет ее отсутствие, поскольку ряд конструкций (например, вложенные тернарные

условные операции и побитовый сдвиг) позволяют опытному профессиональному разработчику написать однострочное выражение, которое, несмотря на высокую производительность в ходе выполнения, может оказаться совершенно не интуитивным с точки зрения синтаксиса для сопровождающего код программиста. Обилие специфических символов для операций с указателями (*, &, ^, ->) лишь усугубляет оговоренную проблему, а отсутствие на данный момент многих современных механизмов (например, так называемого “сборщика мусора”) заставляет программиста тратить дополнительное время на задачи, относящиеся к управлению ресурсами и системному программированию вместо того, чтобы сосредоточиться на прикладной стороне и логике разрабатываемого приложения. Множество других существенных недостатков языка более подробно рассмотрено в работе “C++ FQA Lite”.

Python

Python - высокоуровневый язык общего назначения, развивающийся и в настоящее время (последняя версия языка 3.5.2 выпущена в июне 2016 года). На момент написания главы он всего лишь на одну позицию отстает от языка C++ по рейтингу TIOBE. Его популярность в научных кругах обеспечила появление ряда инструментов и дополнений широкого профиля, в том числе расширяющих возможности для исследований, а также выполнения научных и инженерных расчетов (например, такие библиотеки, как SciPy и NumPy). “The Zen of Python” (“Дзен Python”) определяет основную философию языка и общие принципы, на которых обязан строиться процесс разработки. Эти принципы содержат ряд высказываний, напрямую выражающих требования, предъявленные ранее в качестве критериев для отбора языков, среди них простота (“Simple is better than

complex”) и удобочитаемость (“Readability counts”). Программы, написанные на Python, легко поддерживать за счет синтаксиса, построенного на принципах сопровождаемости. Перечисленные особенности языка делают заведомо трудным написание программ низкого качества.

Python легко установить на различные операционные системы. Поддерживается установка на основе сборки исходного кода.

Python и C++; Cython

Python в чистом виде не предоставляет разрабатываемым программам такие качества, как независимость, компилируемость и высокую производительность. Но в составе индекса разработанных библиотек и расширений (“PyPI” - “Python Package Index”) присутствует компилятор Cython, позволяющий преобразовывать программу Python в оптимизированный код на языке C++, что косвенно гарантирует компилируемость и независимость, свойственные языку C++, а также высокий уровень оптимизации. Результаты сравнений показали, что код, полученный с помощью Cython, приближается по производительности к такому же коду, написанному изначально на C++, в несколько раз превосходя по этому же параметру аналогичный код, выполненный с применением стандартного интерпретатора Python.

Таким образом, использование языка Python с последующей оптимизацией и доработкой кода на уровне языка C++ предоставляет разработчику естественный синтез достоинств обоих языков. Следует заметить, что и среди отзывов ряда других ученых, а также промышленных и индивидуальных программистов, воспользовавшихся компилятором Cython, отмечались высокая производительность приложений, оптимизация

вычислительных процессов, а также общее улучшение качества сопровождаемости исходного кода.

Go

Язык Go активно поддерживается как корпорацией Google, так и сообществом, непрерывно развиваясь и в настоящее время. Принцип простоты является ключевым для Go, а легкая система создания подключаемых пакетов позволяет безгранично расширять уже имеющиеся возможности языка.

Язык быстро обрел последователей в силу кроссплатформенности, универсальности, а также понятного синтаксиса, поверхностно схожего с C, но более строгого для обеспечения удобочитаемости. Одним из самых востребованных принципов языка является современность и ориентированность на задачи, актуальные для XXI века. Наиболее актуальные решения (в том числе автоматизация работы серверов, а также быстрое создание и развертывание веб-приложений) можно организовать в несколько строчек кода за счет большого количества разнообразных функций в стандартной библиотеке.

Язык представлен набором инструментов, позволяющих легко компилировать и тестировать разрабатываемый программный продукт, а также предоставляющих легкий доступ к справочным материалам, включая локальный тур по языку для быстрого постижения его основ. Важно заметить, что разрабатываемая программа не требует никакой внешней библиотеки или прослойки для реализации всех функций, предоставляемых ей языком, что гарантирует независимость конечного приложения.

Среди принципиальных нововведений языка Go можно отметить инструкцию defer, позволяющую выполнять освобождение запрошенных у системы ресурсов функцией post factum, а также новую концепцию механизма обработки исключительных ситуаций. Особое внимание необходимо уделить одной особенности, способной в несколько раз увеличить быстродействие написанной на этом языке программы, обеспечивая крайне высокую производительность последней - механизм управления Go-подпрограммами (“Go-routines”). Go-подпрограммой называется независимо вызываемая функция, обеспечивающая параллельное выполнение нескольких задач в рамках одного приложения.

Go-подпрограммы отличаются от потоков операционной системы, несмотря на схожесть выполняемых задач.

Размер стека, который операционная система выделяет для рабочей области потока, представляет собой фиксированное значение, которое может не соответствовать нуждам задачи, выполняющейся в рамках означенного потока. Поэтому, как правило, стек заведомо выделяется достаточно большим, что косвенно решает проблему, но в то же время существенно ограничивает количество поддерживаемых потоков как в рамках одного приложения, так и в рамках системы в целом. Механизм же, который использует Go, подразумевает использование динамического стека, размер которого изначально небольшой, однако, может изменяться при необходимости, что позволяет поддерживать несколько сотен тысяч go-подпрограмм, а также обеспечивает большую глубину рекурсии, если это необходимо для поставленной в рамках приложения задачи.

Процесс перехода от одного потока к другому осуществляется с помощью специальной функции ядра операционной системы по аппаратному

прерыванию и требует переключения контекста - то есть набора значений регистров, указателей на области памяти и других параметров, определяющих состояние конкретного потока. С точки зрения производительности такая процедура обходится очень дорого, не в последнюю очередь, из-за большого количества обращений к памяти, причем не локализованной относительно основной программы, которой принадлежат потоки, а также того факта, что контекст, определяемый для конкретного потока, принадлежит исключительно ему, что не позволяет сократить требуемое количество переключений между контекстами.

Язык Go предоставляет разрабатываемой его средствами программе собственную подсистему планирования, позволяющую распределять выполнение нескольких Go-подпрограмм на множестве потоков операционной системы. Вызов такой подсистемы планирования осуществляется не вследствие аппаратного прерывания, а программно, по необходимости, в процессе выполнения какой-либо из соответствующих инструкций (к примеру, при работе с функциями синхронизации Go-подпрограмм). Указанный принцип не требует переключения контекстов и заведомо локализован относительно программы Go, при этом сам процесс планирования работает по тому же принципу, что и аналогичная функция в ядре операционной системы. Таким образом, переключение между go-подпрограммами осуществляется быстрее, чем между стандартными потоками.

Итоги; принцип межпроцессного взаимодействия

Мощные инструменты для поддержки параллельного программирования представляют сильную сторону Go, и математический аппарат разработанной модели было решено реализовать только на этом

языке, в то время как графическое представление данных, а также основные операции ввода-вывода и взаимодействие с пользовательским интерфейсом, включая средства редактирования входных данных для запуска описываемой модели, выполняет программа, написанная на двух языках: Python и C++. Для осуществления взаимосвязи между двумя составляющими конечной программной реализации модели используется текстовый формат обмена данными JSON (“JavaScript Object Notation”) в силу его открытости, отсутствия необходимости в принудительной подготовке структур к сериализации, компактности и отсутствия потребности в избыточных символах для корректного представления описываемой структуры. Функции для работы с данным форматом поддерживаются на уровне стандартной библиотеки как в Python, так и в Go.

Вызов внешнего процесса зачастую бывает неэффективным с точки зрения производительности и наделен потенциальными изъянами в области безопасности; однако, Python предоставляет возможность осуществлять подобный вызов достаточно быстро и безопасно за счет модуля subprocess, позволяющего по умолчанию не вызывать процесс через оболочку операционной системы, тем самым сохраняя время и избавляя от потенциальной бреши в безопасности конечного приложения. Поэтому нет никаких причин опасаться за этот вид взаимодействия в рамках данной программной реализации.

Описание взаимосвязей между примененными в разработке технологиями

Необходимость в использовании разнообразных языков, библиотек, компиляторов и технологий для достижения поставленной в рамках разрабатываемого приложения задачи потребовала, прежде всего, определить

систему задействованных для корректного построения проекта взаимосвязей, схематически представленных на рисунке 20.

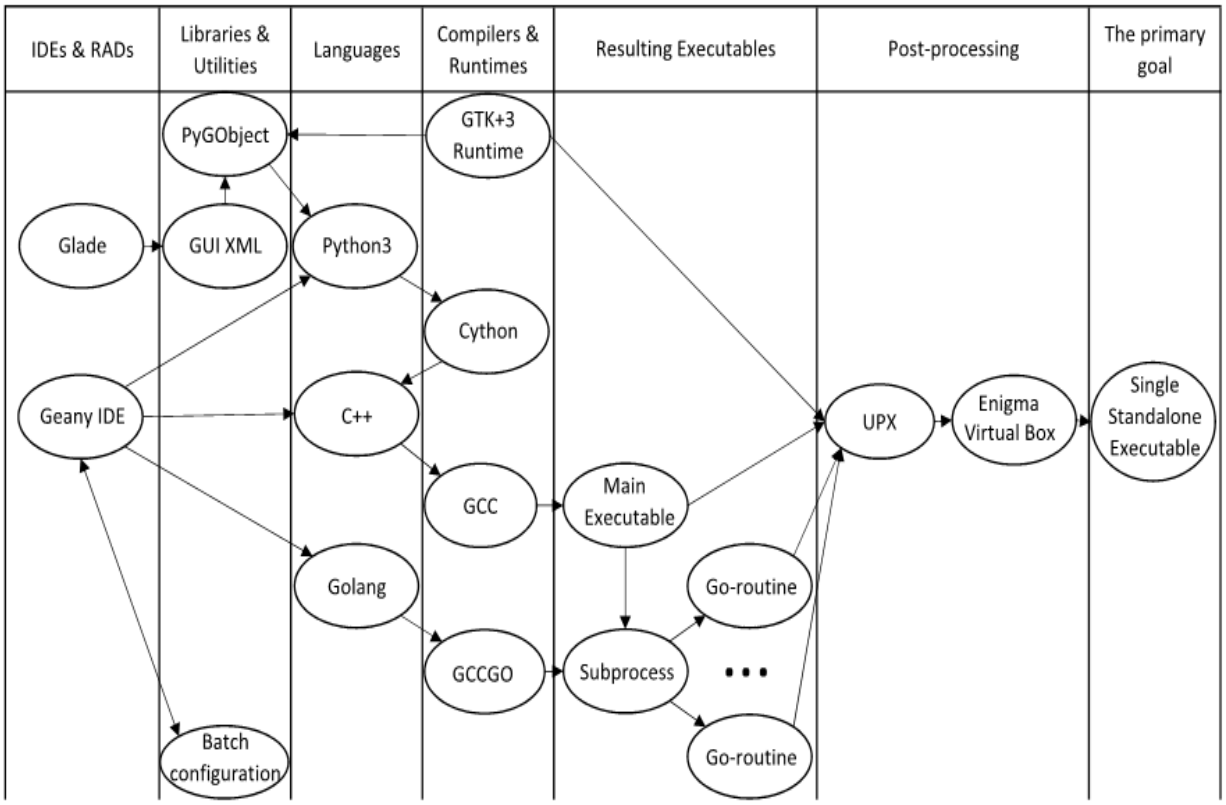

Рис. 20. Взаимосвязь различных слоев технологий для реализации поставленной задачи (в случае целевой платформы Windows).

В качестве кроссплатформенной графической библиотеки, позволяющей разработать гибкий, удобный и современный интерфейс пользователя без добавления избыточного функционала, выбрана GTK+ версии 3. Для доступа к среде выполнения GTK и обеспечения связи между данной библиотекой и реализованным на Python приложением был использован модуль расширения PyGObject. Для создания интерфейса приложения на базе GTK существует Glade, средство быстрой разработки приложений (RAD), которое позволяет создать файл в нотации XML для автоматической загрузки графического интерфейса пользователя (GUI) и

последующего доступа к отдельным компонентам этого интерфейса во время выполнения программы.

Для написания непосредственно программного кода используется интегрированная среда разработки (IDE) Geany, представляющая собой расширенный блокнот с подсветкой синтаксиса, областью вывода результатов сборки проекта, а также основным функционалом для осуществления упомянутой сборки. Geany легко распознает синтаксис и особенности таких языков, как Python, C++, Go и других. Также эта среда позволяет легко редактировать команды, требуемые для проведения корректной сборки, адаптируя их под нужды программиста. Существует возможность подключения и непосредственного редактирования скриптов для терминала (Shell Script), позволяющих более тонко задавать конфигурацию среды. При этом Geany, как и Glade, является свободной средой, не требует лишних ресурсов и может легко выполняться на базе аппаратного обеспечения не очень производительных, по современным меркам, вычислительных машин.

Разработанный интерфейс внедряется в код программы, написанной на языке Python версии 3. Затем, с помощью Cython, этот код преобразуется в аналогичный для C++ и оптимизируется - частично вручную, частично за счет задания соответствующих опций на этапе компиляции. В данном случае, в роли компилятора выступает GCC (“GNU Compiler Collection”, “набор компиляторов GNU”), кроссплатформенный свободный компилятор, автоматически включаемый в большинство современных дистрибутивов Linux (так же, как и GTK+).

Полученная основная программа осуществляет вызов отдельно собранной реализации математического аппарата для проведения требуемых

вычислений. Программа, отвечающая за обеспечение данного аппарата, написана на языке Golang и скомпилирована его же средствами в качестве вызываемого подпроцесса относительно основной программы, написанной на языках Python и C++. При необходимости, вызываемый подпроцесс может разбиваться на ряд Go-подпрограмм для обеспечения задач параллельного программирования с целью улучшения быстродействия представляемого математического аппарата. Дополнительно, для улучшения качества работы вызываемого подпроцесса, на этапе его компиляции подключаются дополнительные опции, обеспечивающие повышенную оптимизацию выходного исполняемого файла. Следует заметить, что средствами Go для операционной системы Linux есть возможность создать динамическую библиотеку, тогда как для Windows в настоящее время доступна только опция вызываемого подпроцесса.

Собранные библиотеки времени выполнения GTK+ и полученные исполняемые файлы представляют собой основу для запуска модели. Для уменьшения размера тех выходных файлов, которые в этом нуждаются, применяется специальный упаковщик UPX (“Ultimate Packer for eXecutables”, “Абсолютный упаковщик исполняемых файлов”), поддерживающий как динамические библиотеки, так и обычные исполняемые файлы для любой современной платформы и любой разрядности. Упаковщик предоставляет высокий уровень сжатия и быстрый высокопроизводительный алгоритм распаковки непосредственно в процессе вызова исполняемого файла.

Для того, чтобы не нарушать лицензию, файлы, обеспечивающие работу библиотеки времени выполнения GTK+ предоставляются отдельно в архиве дистрибутива. Но сама основа для запуска модели при этом объединяется в единый исполняемый файл, обеспечивая большую

мобильность для приложения относительно целевой программно-аппаратной среды и отсутствие необходимости в прохождении сложной процедуры установки для развертывания приложения и запуска модели. Для создания единого исполняемого файла используется свободное приложение Enigma Virtual Box, обеспечивающее виртуализацию файловой системы. Для того, чтобы гарантировать правильную последовательность вызовов в случае потенциальных конфликтов между составными частями проекта, используется дополнительно написанный процесс диспетчеризации внутренних вызовов исполняемых файлов. С точки зрения самих компонентов конечного приложения, обеспечивающего корректное выполнение модели, упомянутый процесс не распознается отдельно от функций виртуализации. Результат взаимодействия всех описанных выше средств и технологий представляет собой программную реализацию модели для поставленной задачи.

Демонстрация возможностей программной реализации модели

Внешний вид основного автономного приложения (в русской локализации), позволяющего запустить многоагентную систему и задать ее параметры, представлен на рисунке 21.

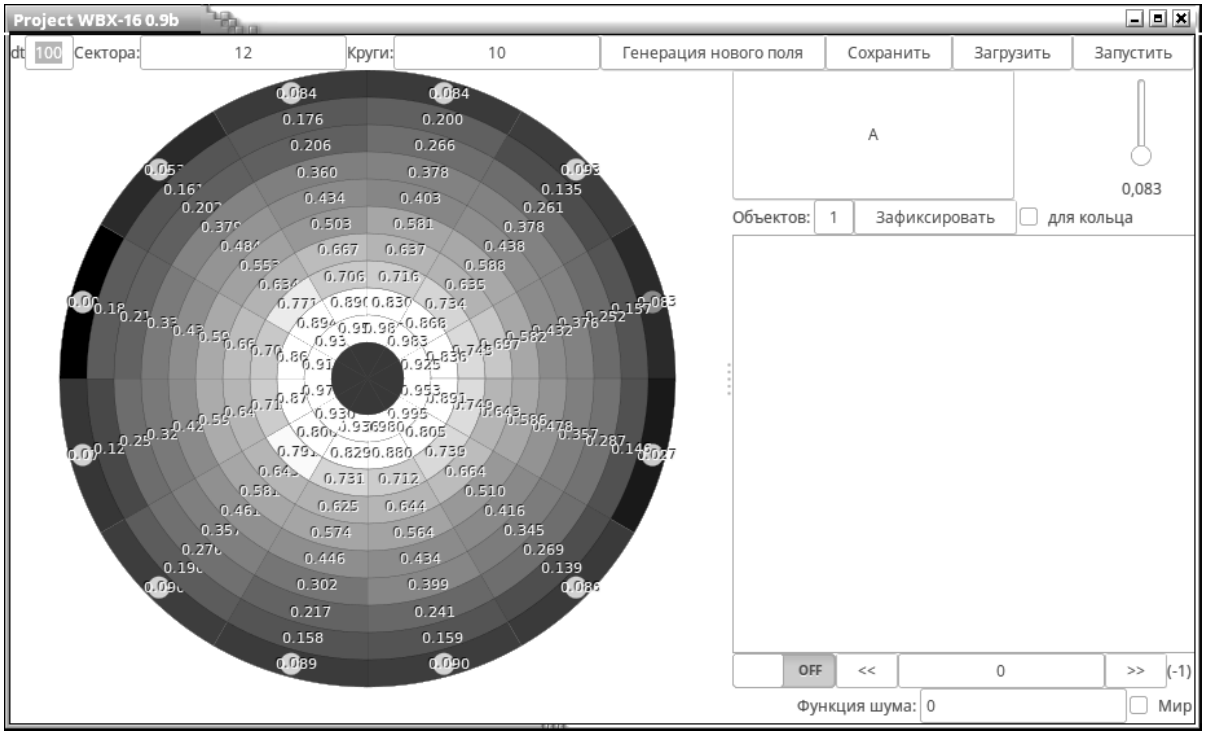

Рис. 21. Скриншот запущенной программной реализации модели в автономном режиме.

Верхняя строка задает основные параметры модели (длительность временного шага, количество секторов и кругов) и позволяет ее пересоздать либо запустить (в двух этих случаях вызывается подпроцесс, обслуживающий математический аппарат), экспортировать модель во внешний файл (в формате JSON) либо импортировать из него. Нижняя строка задает функцию шума (строку, состоящую из нулей и единиц, которая определяет последовательность ввода помех на каждом шаге модели в коммуникацию агентов) и позволяет включить изображение участка карты мира вместо вероятностно-численного представления секторов игрового поля.

Справа представлена панель настроек, позволяющая переключаться между картами осуществимостей (A) и уязвимостей (B) для текущего

игрового поля. Специальный бегунок позволяет изменять значение соответствующей вероятности для выделенного на игровом поле сектора (выделение осуществляется посредством предварительного щелчка левой кнопкой мыши по изображению требуемого сектора). На второй строке панели расположены инструменты для задания количества размещаемых агентов в соответствующем секторе или кольце (кольцо определяется по принадлежности выделенного сектора к нему при активации соответствующего флажка). Область, представленная ниже, содержит необходимые записи о состоянии модели в ходе работы проигрывателя, который из соображений удобства использования располагается непосредственно под этой областью. Проигрыватель состоит из выключателя, кнопок перемотки, окошечка с указанием текущего проигрываемого шага и кодового номера, позволяющего быстро определить общий исход сражения (-1 - проигрыватель выключен, 1 - одиночная победа агента, 2 - поражение агентов, 3 - коллективная победа агентов, 0 - зарезервированный код для отладки в случае неопределенного ошибочного исхода).

Само поле находится с левой стороны и представляет собой ряд окружностей, разделенных на сектора. Цвет каждого сектора определяет соответствующее значение вероятности для карты осуществимостей либо уязвимостей по градации от черного (нуль) к белому (единица). Цель отображается в середине в виде затемненного круга, а агенты показаны как белые, частично прозрачные, круги в секторах, соответствующих их расположению.

Программа позволяет легко модифицировать параметры модели, а также просматривать результаты ее выполнения, в том числе в покадровом режиме.

Недостатки первой автономной версии с точки зрения реализуемого программного комплекса программ и описание обновленного решения

Представленная версия приложения не была конечной ввиду ряда ограничений и недостатков, свойственных первой версии. Эти проблемы не представляли затруднений при работе с одной моделью, но становились ощутимыми на уровне комплекса программных продуктов, которые будут представлены в последующих главах.

— Например, программа позволяла работать с данными только через графический интерфейс. Однако, нередко бывает необходимо задавать данные, характеризующие параметры модели и агентов, в табличном виде, минуя графический интерфейс. Это актуально, например, для задачи единовременной обработки множества данных, описывающих конфигурацию системы, в пакетном режиме с целью их классификации (например, разбиения по уровням в реальном времени).
— Программное обеспечение позволяло отобразить иллюстрацию физической карты моделируемой местности и наложить ее на область моделирования. Однако, не представлялось легкого способа извлечения иллюстрации модели, соответствующей конкретной вероятностной карте (осуществимости или уязвимости), а также не представлялось возможным наложить иллюстрацию физической

карты на график вероятностных распределений, поскольку не поддерживалась прозрачность.

— Используется растровая графика, затрудняющая масштабирование. Нет возможности отследить траектории агентов и их местоположение в физических координатах.

— Для того, чтобы просмотреть результат, необходим проигрыватель, входящий в состав приложения. Нет возможности сделать анимацию моделирования переносимой и независимой от средств приложения.

— Также модель в принципе не позволяет определить местоположение объектов с высокой точностью (нет преобразования между координатами модели и физическими координатами).

— Взаимодействие с моделью во время проигрывания ограничено действиями, связанными с управлением воспроизведением (запуском, остановкой, перемоткой и т. п.). Нет поддержки средств, которые могли бы обеспечить тренировочный процесс для пользователя-оператора системы.

Для устранения перечисленных недочетов была разработана новая система формирования данных о модели поверх уже построенной. Разработка представляет собой систему серверных конвейеров, преобразующих Excel-совместимый документ в формате XML с подробным описанием состояний в HTML5 [87] документ с динамически меняющимся автономным изображением в векторном текстовом графическом формате SVG, отражающим изменение состояний модели во времени. Набор таких HTML-документов, в свою очередь, можно переработать в новый XML-

документ, позволяющий в удобной форме отслеживать изменение макропараметров, анализировать их и выбирать оптимальную стратегию боя.

Приложение задает клиент-серверную архитектуру, однако сервер может быть установлен и развернут локально всего одной командой, обеспечивая функционал обычного кроссплатформенного desktop-приложения. Поддерживается Astra Linux.

Для эффективной реализации приложения и ускорения работы серверных компонентов потребовалось применить межъязыковое взаимодействие на базе динамических библиотек и самостоятельно внести некоторые изменения в компилятор языка Go. Общая процедура подробно описана в статье «Программная реализация межъязыкового взаимодействия на базе динамических библиотек». В случае межъязыкового взаимодействия один процесс может переключаться между элементарными операциями, реализованными в виде процедур и функций, причем написанных на разных языках. Все требуемые операции протекают в рамках единого приложения, а единообразие функций обеспечивается сведением «иноязычных» фрагментов кода в библиотеку с последующей компиляцией в машинный код, ориентированный на целевую архитектуру приложения. Особенно эффективен принцип межъязыкового взаимодействия при объединении разных парадигм программирования для решения одной поставленной задачи.

Пример экспортируемого независимого динамического изображения действующей модели в формате SVG показан на рисунке 22. За счет структуры из дополнительных атрибутов, изображение легко представляется в табличном формате для дальнейшей аналитической обработки.

Рис. 22. Экспортируемое динамическое векторное изображение моделируемого боя. Изображение допускает интерактивность и не требует специального проигрывателя (только браузер). Внутри изображения хранятся данные о модели и каждом сделанном шаге. Отображается траектория движения. Допускается управление прозрачностью модели.

Численный метод внешней оптимизации для идентификации марковских процессов

Важной особенностью практического использования марковских моделей применительно к указанным задачам является тот факт, что их параметры идентифицируются по результатам наблюдений, представленных наборами значений исследуемых функций в контрольных точках.

Марковские модели для описания динамики переходов между состояниями представляются ориентированными графами, в которых вершины соответствуют состояниям, а дуги соответствуют переходам, для которых выполняются свойства пуассоновских потоков событий [88]. В этих потоках число событий N, попадающих в любой временной интервал длины τ, начинающийся в момент t, распределено согласно закону Пуассона:

$$P_{t,\tau}(N = r) = \frac{a(t,\tau)^r}{r!} e^{-a(t,\tau)},$$

где $P_{t,\tau}(X = r)$ - вероятность появления r событий в течение рассматриваемого интервала, $a(t,\tau)$ - среднее число событий, попадающих в интервал длины τ, начинающийся в момент времени t. Рассматриваются только стационарные потоки, в которых $a(t,\tau)=\eta\tau$, а $\eta=const$ есть интенсивность стационарного потока. Упомянутые выше предположения о свойствах потоков событий обычны для прикладных задач, так как эти потоки (или потоки, близкие к ним по свойствам) часто встречаются на практике благодаря предельным теоремам для потоков событий.

Полагается, что:

– для указанных процессов с $n+1$ дискретными состояниями и непрерывным временем заданы начальные распределения вероятностей и наблюдаемые частоты пребывания в состояниях процессов $\{F_{id}\}_{i=0,\ldots,n}$ в моменты времени $\{t_d\}_{d=0,\ldots,D-1}$, где D – количество моментов времени, в которые фиксировались частоты $F_{id};\ \ 0 \le t_d \le T$ где T – конечный момент времени;

– интенсивности переходов между состояниями полностью или частично являются неизвестными (свободными) параметрами.

Как было показано ранее, динамика изменения вероятностей пребывания в состояниях процесса определяется системой обыкновенных дифференциальных уравнений Колмогорова в матричной форме. Для указанной системы уравнений ставится задача идентификации набора параметров $\boldsymbol{\lambda_k}$, задающих упорядоченный набор интенсивностей переходов между состояниями.

Значения параметров из этого набора определяются путем сравнения наблюдаемых и прогнозируемых гистограмм, описывающих распределения частот пребывания в состояниях модели, а именно: вычисляются значения, обеспечивающие наилучшее соответствие наблюдаемых и ожидаемых частот попадания в определенное состояние системы в заданные моменты времени.

Для решения этих проблем был предложен численный метод ***перебора значимых параметров***, предназначенный для идентификации набора параметров $\boldsymbol{\lambda}$, определяющих матрицу $\mathbf{M}$. Этот метод [89, 90] представлен далее в версии, совместимой с задачей моделирования прикладных многоагентных систем, упомянутых выше.

Несмотря на то, что скорость оптимизации с применением описанного алгоритма, как правило, близка к приемлемой, существует целый ряд задач, для которых одним из требований является мгновенное принятие решений. В частности, при достаточно большом количестве параметров, обусловленном высокой точностью разбиения игрового пространства на множество колец и секторов в ходе работы вероятностной модели поведения прикладной многоагентной системы, необходимо обеспечить адаптивность алгоритма оптимизации в отношении предустановленных параметров алгоритма, влияющих на скорость его работы, таким образом, чтобы увеличить скорость сходимости без потери качества получаемого в результате работы метода значения критерия. Это обстоятельство и обуславливает необходимость в разработке и применении нового численного метода, позволяющего адаптировать параметры основного метода оптимизации для повышения эффективности его работы, а также продемонстрировать реализацию нового метода в виде алгоритма вычислений и в качестве действующего программного инструмента.

Метод перебора значимых параметров: алгоритм вычислений

1. Используя имеющиеся результаты наблюдений, вычислить в качестве начальных оценок параметров $\lambda_i (i = 0,...,m)$, обозначаемых как $\lambda_i^0 (i = 0,\ldots,m)$, случайно взятые значения интенсивностей переходов между парами соответствующих состояний марковского процесса в единицу времени. Полученные оценки рассматривать как начальные приближения к идентифицируемым параметрам.

2. Положить $j = 1, \quad k = 1, \quad \beta_1 = 1 + \gamma_1$, где γ_1 – параметр алгоритма.

3. Оценить чувствительность критерия X^2 в малой окрестности каждой из текущих оценок идентифицируемых параметров $\boldsymbol{\lambda}_i^j = (i = 0,\ldots,m)$ на j-й итерации алгоритма, используя для этого разностные аппроксимации абсолютных значений частных производных $\delta_i(\varepsilon) = | X^2(\lambda_0^j,\ldots,\lambda_i^j + \varepsilon,\ldots,\lambda_m^j) - X^2(\lambda_0^j,\ldots,\lambda_i^j,\ldots,\lambda_m^j) / \varepsilon |$, где $i = 0,\ldots,m;\ \ \varepsilon$ _ параметр алгоритма.

4. Выбрать l текущих оценок идентифицируемых параметров $\lambda_{i_q}^j$, где $i_q \in \{0,\ldots,m\};\ \ q = 1,\ldots,l;\ \ l$ – параметр алгоритма, соответствующий оценкам, имеющим наибольшие значения оценок чувствительности $\delta_i(\varepsilon)$.

5. Для каждой из выбранных на шаге 4 текущих оценок идентифицируемых параметров $\lambda_{i_q}^j$ на j-й итерации алгоритма, где $i_q \in \{0,\ldots,m\};\ \ q = 1,\ldots,l,$ вычислить верхнюю $\lambda_{i_q}^{j+}$ и нижнюю $\lambda_{i_q}^{j-}$ границу сдвига оценки по формулам: $\lambda_{i_q}^{j+} = \beta_j \lambda_{i_q}^j,\ \ \lambda_{i_q}^{j-} = \beta_j^{-1} \lambda_{i_q}^j$.

6. Выполнив полный перебор всех вариантов текущих оценок набора параметров $\boldsymbol{\lambda}_j = (\lambda_0^j,\ldots,\lambda_m^j)^T$, в которых каждый из его компонентов $\lambda_{i_q}^j$, где $i_q \in \{0,\ldots,m\};\ \ q = 1,\ldots,l,$ принимает только три возможных значения из множества $\{\lambda_{i_q}^{j-1,-}, \lambda_{i_q}^{j-1}, \lambda_{i_q}^{j-1,+}\}$, а оставшиеся компоненты – значение своей текущей оценки на j-й итерации алгоритма, выбрать из указанных вариантов одну из оценок $\boldsymbol{\lambda}_{j,*} = (\lambda_{0,*}^j,\ldots,\lambda_{m,*}^j)^T$, обеспечивающих максимальное значение критерия X^2, равное $X_{j,max}^2$.

7. Если для всех компонентов выбранного набора $\boldsymbol{\lambda}_{j,*}$ выполняется равенство $\lambda_{i,*}^{j} = \lambda_i^{j-1} (i = 0,...,m)$, то положить $k = k+1, \gamma_k = y_{k-1} / 2$.

8. Положить $j = j+1, \quad \beta_j = 1+\gamma_k, \quad \boldsymbol{\lambda}_{\mathbf{j}} = \boldsymbol{\lambda}_{\mathbf{j-1,*}}$.

9. Если $| X^2_{j-1,min} - X^2_* | > \varepsilon_t$, где X^2_*, ε_t – параметры алгоритма, то перейти к шагу 3, иначе завершить вычисления.

Комментарий к представленному алгоритму вычислений

Для решаемых на практике типовых задач общее время вычислений зависит от параметра *l* монотонно, достигая наименьшего значения при *l=1*. Временн*а*я сложность одной итерации модифицированного алгоритма при *l=1* есть *O(m)*, что свидетельствует об асимптотически линейной зависимости времени вычислений, необходимых для выполнения одной итерации, от числа идентифицируемых параметров (рис. 23).

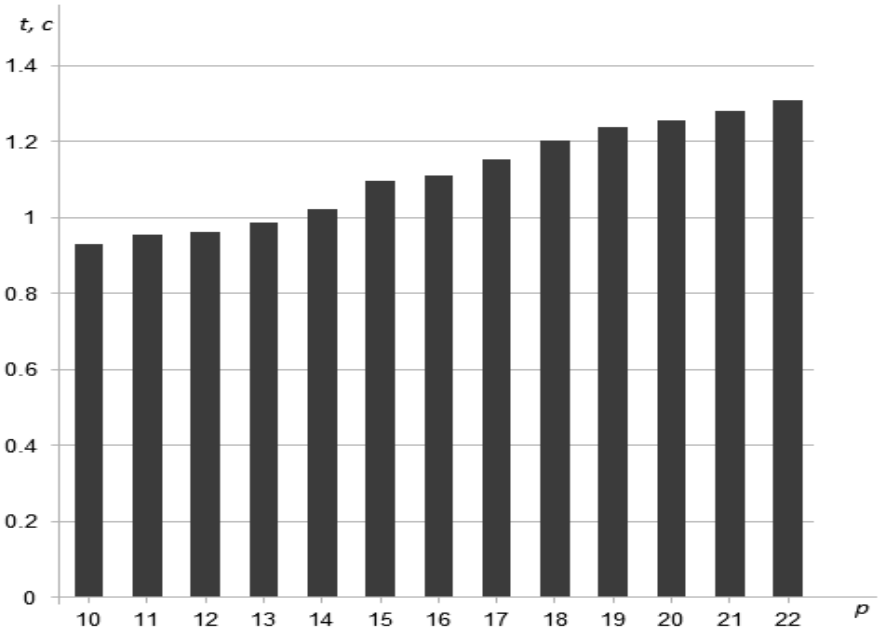

Рис. 23. Эмпирическая зависимость времени выполнения алгоритма (t) от числа параметров (p).

Введение терминологии для нового математического метода

Приведенный выше метод оптимизации (и соответствующий ему алгоритм) будем условно называть «основным».

Параметры, существенно влияющие на качество и время работы основного алгоритма, но не относящиеся к числу оптимизируемых основным методом значимых параметров, будем называть «метапараметрами основного алгоритма».

Эффективность основного алгоритма за одну итерацию можно рассматривать как отношение значения максимизируемого критерия ко времени, затраченному на работу алгоритма, при условии, что исходные значимые параметры остаются неизменными между итерациями (сохраняют свои изначальные значения в качестве входных параметров основного алгоритма). Тогда, адаптивно меняя метапараметры основного алгоритма оптимизации, оказывающие наибольшее влияние на рост его эффективности, можно повысить эффективность основного математического метода в целом.

Эмпирически было определено, что такими метапараметрами являются:

- M – число варьируемых значимых параметров,
- Δ – начальная поправка для значимых параметров.

Алгоритм-надстройку, осуществляющую модификацию метапараметров основного алгоритма с целью увеличения критерия эффективности, будем называть «внешний алгоритм оптимизации».

Для реализации соответствующего численного метода необходимо решить следующие задачи:

- Каким образом будет осуществлено взаимодействие с основным методом?
- Как алгоритмически будет представлен процесс адаптивного последовательного изменения метапараметров?

Взаимодействие между методами: шаблон «декоратор»

Алгоритмический шаблон проектирования «декоратор» описывает программную функцию высшего порядка, которая, на основе получаемых значений аргументов, включая указатель на модифицируемую функцию, преобразует процесс работы последней, внося изменения в ее входные параметры и анализируя (с последующим внесением необходимых изменений) выходные данные.

В отношении описанного выше метода оптимизации можно рассматривать алгоритм метода перебора значимых параметров как модифицируемую функцию, для которой в качестве изменяемых входных данных задаются соответствующие значения изменяемых метапараметров.

Общая иллюстрация способа применения шаблона «декоратор» в ходе вызова декорируемой функции представлена ниже (рис. 24).

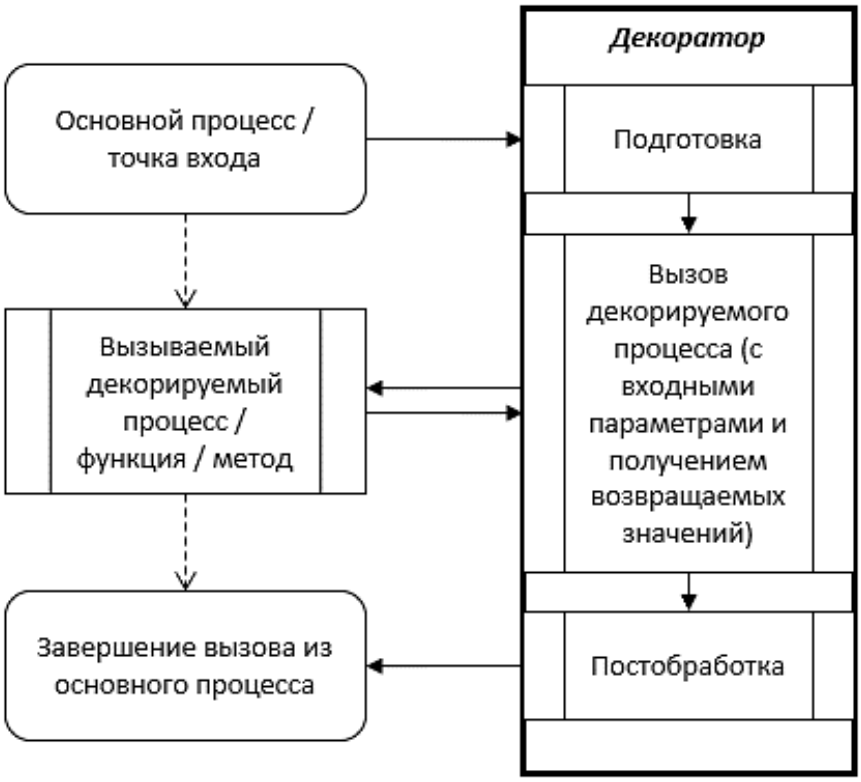

Рис. 24. Общий пример вызова декоратора из произвольного основного процесса. В качестве декорируемого процесса может выступать произвольная функция, метод или внешний процесс, которому временно передается управление из вызывающего процесса.

Условно процесс можно разделить на три части: подготовка к вызову декорируемого процесса, непосредственно вызов и постобработка.

Подготовка заключается в осуществлении действий, предваряющих запуск вызываемого (также называемого «декорируемым») процесса. Это может быть вывод сообщения для отладки, предварительное сохранение значимых данных перед осуществлением транзакции, представленной декорируемым процессом, подготовка промежуточных вычислений на основе входных аргументов и параметров, получаемых процессом локально или из глобального контекста, а также другие процедуры, осуществляющие

инициализацию модифицированного окружения и контекста вызываемого процесса для достижения конкретной поставленной перед декоратором задачи.

Вызов декорируемого процесса может сопровождаться как изменением входных аргументов и возвращаемых значений, так и их сохранением в случае, если воздействие на окружение процесса осуществлено каким-то другим образом (например, через вызов другого процесса в процессе подготовки декоратора).

Постобработка – это процесс модификации выходных данных либо применения их в промежуточных вычислениях для анализа и извлечения результатов с целью подведения итогов, работу декоратора.

Примером автономной программной реализации такого шаблона можно считать приложение, вызывающее другое приложение, библиотечную функцию или системный вызов, предварительно подбирая параметры вызова таким образом, чтобы достичь желаемого результата.

Описание процесса адаптивного изменения метапараметров

Надстройка алгоритма представляет собой самостоятельную функцию, выполненную в форме шаблона проектирования «декоратор» для объектно-ориентированного метода, представляющего собой реализацию алгоритма оптимизации для заданного Марковского процесса либо группы процессов. Среди всех задаваемых аргументов функции последовательно изменяются параметры модели, отвечающие за количество изменяемых параметров и значение начального сдвига параметров.

Представим «сетку» - двумерное пространство, в котором по одной оси с фиксированным шагом h_1 задается количество единовременно изменяемых

параметров от 1 до n, а по другой – также, с фиксированным шагом h_2, определяется начальный сдвиг искомых параметров (от h_2 до 1). Полученное пространство позволяет определить значение критерия эффективности метода в каждой ячейке «сетки» - в зависимости от конкретного значения модифицируемых параметров.

Описываемую сетку можно представить схематическим рисунком, изображенным ниже (рис. 25).

Рис.25. Двумерное пространство параметров для проведения процедуры оптимизации.

Время, затрачиваемое на эмпирический поиск наиболее оптимального решения (например, посредством полного перебора), значительно возрастает с увеличением сложности модели, делая такой поиск непригодным на практике. В связи с этим наблюдением возникает необходимость в численном методе, позволяющем качественно улучшить эффективность основного метода, при этом адаптирующемся под предельно допустимое

время, как выделяемое для самого процесса поиска оптимального решения, так и затрачиваемое на каждый шаг алгоритма оптимизации, для которого осуществляется подбор оптимальных параметров запуска.

В разработанном численном методе задача поиска в рамках заданного пространства параметров выполняется по следующему алгоритму.

Метод внешней оптимизации: алгоритм вычислений

1. Задается минимальное число изменяемых параметров ($n \geq 1$), максимальное число изменяемых параметров ($U \leq$ количеству параметров), шаг сетки по параметрам (S), шаг сетки (D), предельный срок вычислений (H), максимальное время (T).

2. Генерируется сетка параметров M,d (количество изменяемых элементов, начальное значение шага), $n \leq M \leq U$, $0 \leq d \leq 1$, шаг M=S, шаг d=D.

3. На сетке выбирается произвольная начальная позиция M_0, d_0 и инициализируется словарь пути (ассоциативный массив, состоящий из пар «ключ-значение», где каждый ключ служит индексом ассоциативного массива и является уникальным элементом среди других ключей; для указанного словаря пути в качестве ключа выступает пара кординат на сетке параметров) Z и массив оптимального результата R, хранящий данные о координатах и значениях словаря пути Z.

4. В окрестностях M_0, d_0 выбирается множество точек $\{M_x, d_y\}$, такое, что каждая координата Mx принадлежит отрезку $[M_{0-1}..M_{0+1}]$, каждая координата d_y принадлежит отрезку $[d_{0-1}..d_{0+1}]$.

5. Инициализируется массив результатов измерений P для хранения вещественных значений с количеством элементов, соответствующим количеству точек во множестве $\{M_x, d_y\}$.

6. Для каждой точки, принадлежащей множеству $\{M_x, d_y\}$, выполняется следующая последовательность действий (6.1-6.9):

6.1. Если координата выбираемой точки принадлежит диапазону соответствующих значений, определенных на шаге 1, то перейти к шагу 6.4.

6.2. Если координата выбираемой точки имеет значение меньше нуля, то присвоить такой координате значение, равное максимальному значению из допустимого диапазона; повторить для каждой координаты.

6.3. Если координата выбираемой точки имеет значение больше максимально допустимого для соответствующего диапазона, то присвоить такой координате значение, равное минимальному значению из допустимого диапазона; повторить для каждой координаты.

6.4. Если точка уже была занесена в словарь пути Z в качестве ключа, то считать значение с соответствующим ключом из словаря и перейти к шагу 6.8.

6.5. Используя имеющийся основной алгоритм оптимизации с параметрами, соответствующими текущему положению сетки в заданной точке, выполняется процедура оптимизации до критерия.

6.6. Время, потраченное на выполнение алгоритма в пункте 5.2, а также достигнутый уровень значения критерия в ходе выполнения алгоритма фиксируются в словаре пути Z, где в качестве ключа словаря используется искомая точка.

6.7. Потраченное время прибавляется к общему времени H0, затраченному на выполнение алгоритма оптимизации во всех искомых точках.

6.8. Занести значение критерия для текущих координат в массив результатов измерений P для соответствующего значения координат заданной точки.

6.9. Если все точки из множества $\{M_x, d_y\}$ перебраны, то переход на шаг 7; иначе – переход на шаг 6.1.

7. Осуществляется поиск наибольшего значения критерия в массиве результатов измерений P и вычисляются соответствующие данному значению координаты M_{max}, d_{max}.

8. Для значения координат M_{max} и d_{max}, используемых в качестве ключей словаря пути Z, извлекаются соответствующие значения затраченного времени и уровня критерия; если затраченное время превышает порог максимального времени T, отводимого на один прогон алгоритма оптимизации с параметрами, то переход на шаг 11.

9. Внешний критерий определяется как функция f от затраченного времени T_o на выполнение вложенного алгоритма оптимизации и значение полученного критерия C_o: $f(C_o, T_o)=C_o/T_o$

10. Если массив оптимального результата R пуст или значение найденного внешнего критерия в массиве R не превышает соответствующего значения критерия для словаря пути Z с ключом (M_{max}, d_{max}), то новый набор значений массива R задается эквивалентным совокупности ключей словаря пути Z, и соответствующему значению внешнего критерия для координат M_{max} и d_{max}.

11. Если $M_{max}==M_0$ и $d_{max}==d_0$, то переход на шаг 12, иначе переход на шаг 15.

12. Задаются случайные координаты сетки M_r и d_r.

13. Если словарь пути Z содержит в качестве ключа координаты сетки M_r и d_r, а размерность пространства ключей словаря не превышает значений соответствующих параметров, заданных на шаге 1, то перейти на шаг 12.

14. Присвоить $M_0=M_r$, $d_0=d_r$.

15. Если время H_0, затраченное на выполнение алгоритма оптимизации на всем ансамбле измеренных параметров, превышает пороговое значение предельного срока вычислений H, то перейти на шаг 16, иначе перейти на шаг 4.

16. Если массив оптимального результата R пуст, то перейти на 18.

17. Вернуть в качестве результата выполнения алгоритма массив оптимального результата R и перейти на 19.

18. Вернуть внутренний код сообщения об ошибке “Начальные параметры заданы неверно”.

19. Завершение работы алгоритма.

Комментарий к алгоритму

В терминах шаблона проектирования «декоратор» описанный выше алгоритм можно разбить на три части, подробно описанные выше: подготовка (шаги 1-6.4), вызов декорируемого процесса (шаг 6.5) и постобработка (шаги 6.6-19).

Подготовка перед запуском декорируемого процесса – основного метода оптимизации – заключается в инициализации сетки допустимых значений изменяемых параметров декорируемого процесса перед его вызовом на основе параметров для процесса-декоратора, описывающего «внешний» метод оптимизации. Инициализируются служебные структуры хранения данных для фиксации траектории движения внешнего метода оптимизации по сетке параметров с целью достижения оптимального результата работы вызываемого процесса на основе принципов, представленных ранее. Запускается цикл с постусловием, позволяющий осуществлять проход по пространству возможных значений параметров, которое ограничено узлами предварительно сгенерированной сетки. Устанавливается количество единовременно изменяемых параметров и начальный сдвиг искомых параметров в соответствии с условиями, определенными для текущей итерации. Также выполняются действия, программная реализация которых варьируется в зависимости от среды выполнения (архитектуры вычислительной системы, операционной системы и т.п.), суть которых сводится к обеспечению работы таймера с целью проведения измерения временного интервала, который необходим для работы основного метода оптимизации при заданных параметрах.

После проведения этих действий производится вызов декорируемого процесса, которым является основной метод оптимизации. Эта процедура необходима для получения значения возвращаемого критерия при заданных параметрах и возможности произвести измерение временного интервала, в ходе которого осуществляется искомый вызов.

Постобработка заключается в фиксации значений, полученных в качестве результата вызова декорируемого процесса с целью последующего

отбора наилучших результатов в ходе работы цикла, а также определения критериев останова алгоритма. Также в ходе постобработки фиксируются возможные ошибки при задании начальных параметров работы внешнего алгоритма оптимизации, который сообщает пользователю о недостижимости поставленной цели при текущих условиях выполнения.

Данный метод может быть применен для улучшения представленного метода оптимизации и может быть расширен для n-мерного пространства параметров за счет увеличения числа шагов, обеспечивающих поиск вокруг текущей точки.

Пример практического применения

Ход запуска и работы программной реализации метода на практическом примере показан на рисунке 26.

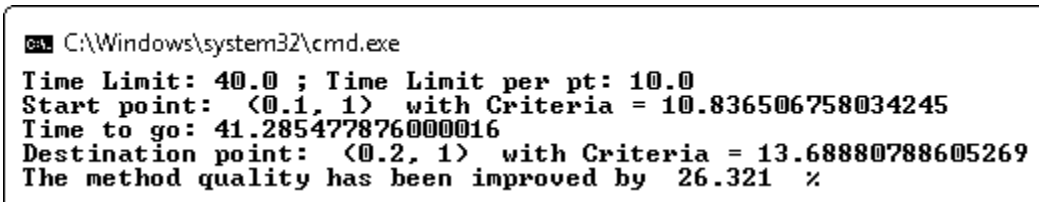

Рис. 26. Пример программной реализации
(основная кроссплатформенная версия без графического интерфейса).

Задается предел времени (в секундах) для всех итераций поиска и для каждого шага. На основе этого, согласно описанному ранее алгоритму, производится поиск оптимального решения. Значение оцениваемого критерия и затраченное время выполнения демонстрируется в соответствующем окне консоли. В частности, при переходе от произвольно выбранной начальной точки в пространстве «сетки» метапараметров (сдвиг = 0,1, количество параметров = 1) к точке (сдвиг = 0,2, количество параметров

= 1) критерий эффективности увеличивается с 10,84 до 13,69 (на 26%), при этом время поиска составило 41,3 с (при увеличении лимита допустимого времени качество может увеличиться). Метод дает гарантию, что основная процедура оптимизации при найденных метапараметрах для начальных условий, в которых проводилась оптимизация, не превысит 10 секунд (на практике полученное время одной итерации в среднем не превышает 0,06 секунд).

Конечная и начальная точка пути поиска, наравне с показателем эффективности, выраженном в процентах, представлена выводом консольного окна и может быть применена при запуске программной реализации модели путем перенаправления стандартного канала вывода в файл или другое хранилище данных, совместимое с форматом данных, возвращаемых методом.

Применение разработанного метода в качестве надстройки над основным методом перебора значимых параметров увеличивает эффективность последнего на 26,32%, что является значимым результатом с точки зрения увеличения общей эффективности решения ввиду того, что основной метод применяется не один раз, а на каждом шаге рассматриваемой прикладной задачи в ходе работы вероятностной модели поведения прикладной многоагентной системы.

При необходимости, может быть осуществлена сериализация промежуточных данных в формат XML, совместимый с большинством современных редакторов электронных таблиц (LibreOffice Calc, Microsoft Excel, WPS Spreadsheets и др.), что позволяет осуществлять необходимые расчеты и анализировать промежуточные результаты поиска оптимального пути по сетке.

Например, можно оценить среднее значение критерия в зависимости от роста изменяемого параметра алгоритма метода – например, начальный сдвиг (рис. 27) или относительное значение времени, затрачиваемого на поиск, при увеличении числа параметров (рис. 28).

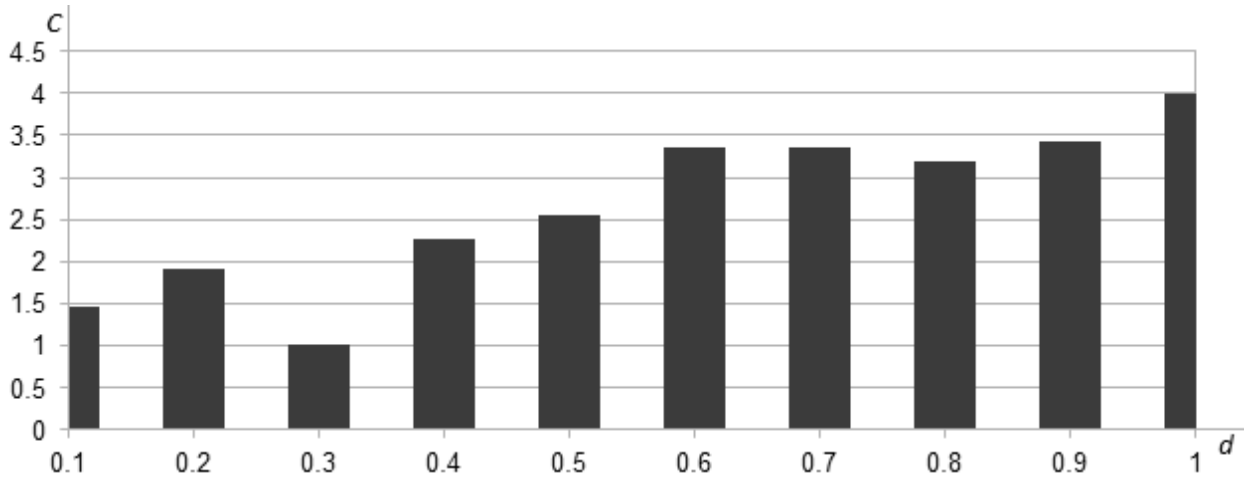

Рис. 27. Среднее значение критерия (c) в зависимости от роста изменяемого параметра алгоритма (d).

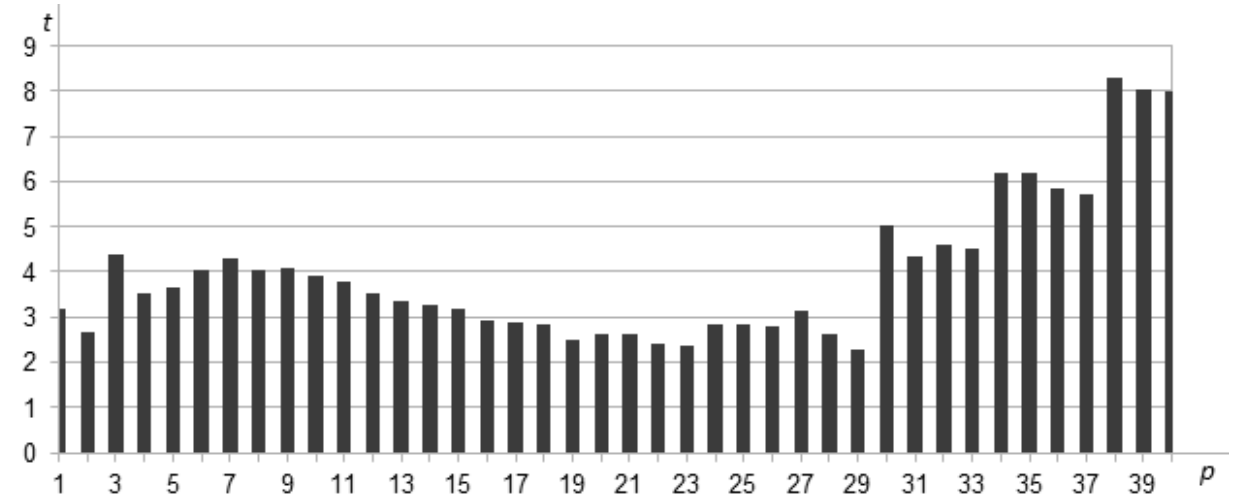

Рис. 28. Относительное значение времени (t), затрачиваемого на поиск, при увеличении числа параметров (p).

Данные могут быть получены для произвольного алгоритма оптимизации. На основе проведенных исследований можно сделать вывод, что представленный математический метод способен повысить эффективность работы произвольного алгоритма оптимизации за счет адаптивного изменения его метапараметров.

Применение и разработка тренажера для автоматизированных беспилотных летательных аппаратов и робототехнических комплексов на базе вероятностной модели поведения прикладной многоагентной системы

В предыдущих главах была представлена разработанная вероятностная модель поведения прикладной многоагентной системы, а также ее программная реализация, позволявшая осуществлять симуляцию игрового взаимодействия множества взаимосвязанных агентов и поражаемой ими цели. Было выдвинуто и доказано предположение, что, используя указанную модель, можно осуществлять прогноз игровых ситуаций с целью выявления наиболее выгодной стратегии, исходя из описания текущего состояния системы и ее участников (агентов и цели). Для этого строилась база игровых состояний путем многократного запуска многоагентных систем с изменением ряда основных параметров, но с сохраненной геометрией игрового мира. После этой процедуры на основе полученной базы проводился анализ данных с целью выявления зависимостей между игровыми состояниями с помощью эмпирически определенных макропараметров, влияющих на качественные характеристики боя и состояния агентов в процессе работы модели [91-93].

Разработанные математические и программные средства предназначены для обеспечения оптимального управления наземными и воздушными комплексами боевой техники в случае ведения боя с эффектом неопределенности, когда перемещение боевой единицы носит намеренно стохастический характер с целью уменьшения вероятности ее поражения целью, выступающей в роли противника. До этого момента упор в работе

был на решение актуальной задачи создания полностью автоматизированных боевых комплексов, необходимость использования которых выявил опыт военных конфликтов последних лет. Тем не менее, может возникнуть ситуация, когда человек должен взять на себя управление в качестве оператора сложной многоагентной системы со стороны цели, обеспечивая ее защиту от потенциального нападения агентов, участвующих на противоположной стороне конфликта в моделируемой ситуации.

В представленных выше материалах не наблюдалось компонента человеко-машинного взаимодействия, который позволил бы вести обучение и определять уровень компетентности потенциальных операторов многоагентной системы, тогда как реализация такого рода обучающего процесса с практической точки зрения полезна тем, что позволила бы избежать выхода из строя реального дорогостоящего оборудования во время тренировки.

Были рассмотрены аналогичные системы, работа которых могла бы обеспечить функционирование описанного процесса обучения. Для некоторых из них характерно наличие математической модели, описывающей адаптивное обучение, однако ограничения реализации в рамках конкретной предметной области не позволяют организовать процесс обучения для оператора сложных систем в общем виде [94-96]. Для других систем не предусмотрено наличие специализированной математической модели для оценки уровня обучения по заданным критериям, несмотря на адаптацию к предметной области сложных систем [97, 98].

С учетом недостатков исследованных подходов к решению поставленной выше задачи был разработан и программно реализован виртуальный адаптивный тренажер на основе модели многоагентной

системы в качестве программного комплекса взаимосвязанных приложений и библиотек.

Основные принципы работы виртуального адаптивного тренажера

Так же, как и для основной модели, игровое поле многагентной системы представлено в виде круглой области, однако в центре находится игрок, играющий роль цели, тогда как агенты постепенно приближаются к нему, стремясь поразить [99-101]. Игрок может переключать текущий вид игрового поля, чтобы изучить карту уязвимостей либо осуществимостей – при этом поле приобретает градиент, соответствующий выбранному режиму. Задача игрока в рамках одной сессии – поразить всех агентов, имитируя цель. Приказ вести огонь на поражение и выбор сектора для ведения боя осуществляется с помощью доступного средства ввода (например, мыши или клавиатуры).

Тренировка осуществляется на наборе из сессий, формируемых из числа генерируемых игровых ситуаций разного уровня сложности, причем «средний» уровень сложности является начальным. Задача игрока в рамках набора сессий – не допускать ошибок, которые могут привести к недостижимости успешного выполнения задачи в каждой отдельно взятой сессии.

Тренировка осуществляется в реальном времени, а скорость течения времени характеризуется параметром, задающим минимальное количество тактов в единицу времени.

Представление игрового поля после запуска модели в режиме тренажера изображено на рис. 29.

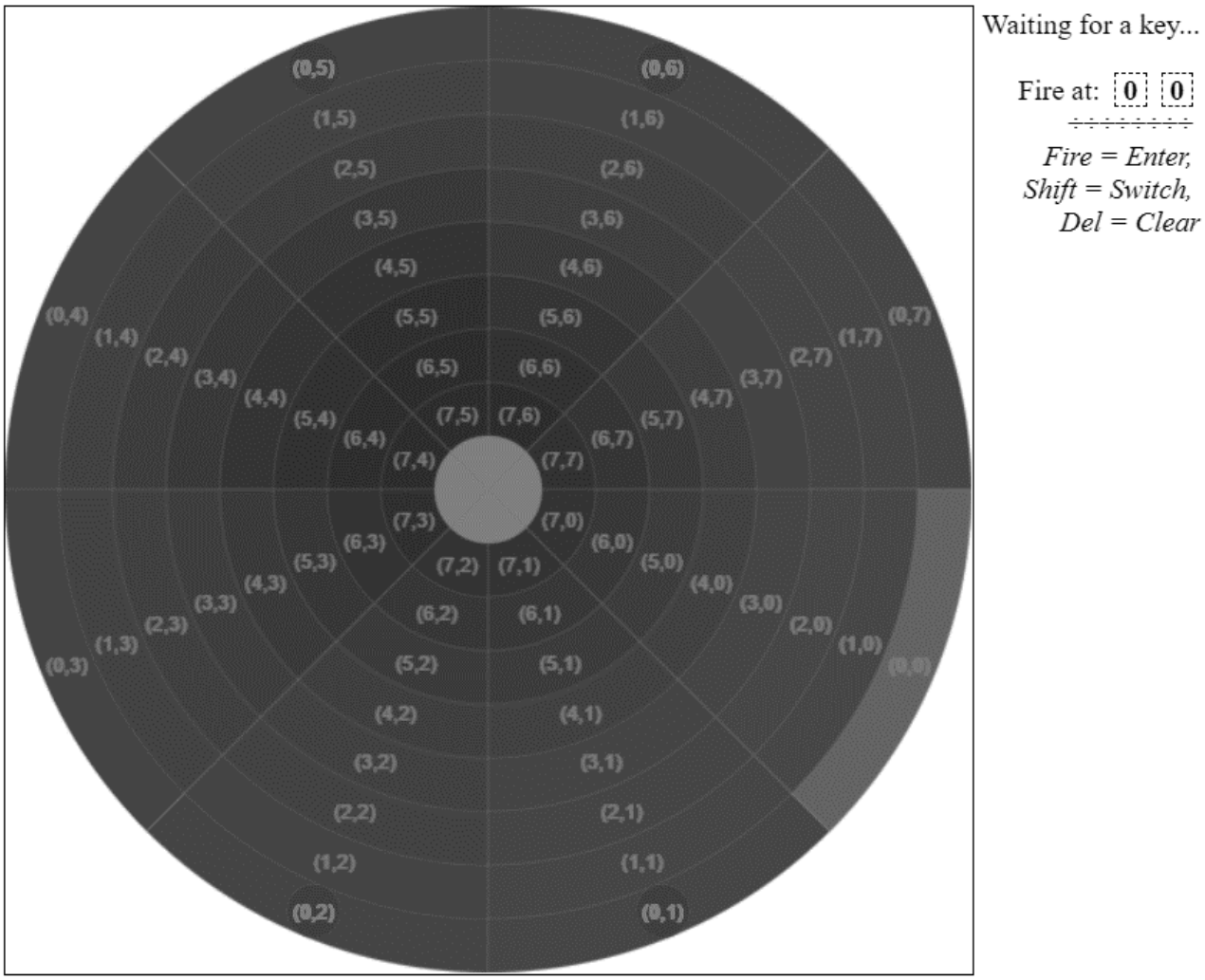

Рис. 29. Игровое поле тренажера (международная кроссплатформенная версия).

Принцип работы тренажера основан на Марковской модели гибели и размножения, то есть является Марковским процессом с постоянной интенсивностью переходов, представляющим переходы испытуемого от уровня 0 до уровня M-1, где M – количество доступных тренажеру уровней. Уровень позволяет условно оценить степень компетентности пользователя – оператора, осуществляющего управление тренажером. Предполагается, что при выполнении поставленной задачи значение текущего уровня для

оператора рано или поздно (в бесконечности) сойдется к значению, соответствующему приближенной оценке его способностей.

Подбор уровней мог бы осуществляться вручную; однако, разработанный ранее механизм прогнозирования благоприятной стратегической ситуации с помощью макропараметров позволяет значительно упростить данную процедуру: ранее было эмпирически показано [102-104], что для рассматриваемой многоагентной системы в качестве макропараметров, выражающих наиболее характерные черты игры, влияющие на ее исход и прогнозируемую выигрышную стратегию, удобно использовать:

T_{win} – время до выигрыша в тактах,

L_{lost} – относительное количество поражённых агентов в конце игры,

N_r – номер ближайшего к цели кольца, в котором располагается агент,

P_a – количество агентов, располагающихся на игровом поле.

Для обеспечения независимости прогнозируемого результата игры от конкретного вида игрового поля, первый и четвёртый из указанных параметров рассматриваются в отношении к числу колец.

Если взять макропараметры T и L, такие, что для 0-ого уровня компетентности оператора их значения составляют, соответственно, ∞ и 100%, для уровня M-1 – Min(T) и 0%, а для уровней 1..M-2 значения указанных макропараметров распределены линейно, то полученные значения N и P для соответствующих параметров T и L отражают ситуацию на игровом поле, определяющую состояние соответствующей сессии при заданном значении уровня оператора. Число используемых уровней определяет точность прогнозирования компетентности оператора. При

необходимости, уровни можно редактировать вручную с помощью специального редактора, входящего в состав комплекса программ.

Описание принципов программной реализации тренажера, подходов к интеграции в различные вычислительные системы и характера взаимодействия с конечным пользователем

Комплекс программ, обеспечивающий работу тренажера, представляет собой кроссплатформенное приложение с графическим кросс-браузерным интерфейсом на базе языка разметки гипертекста с поддержкой стандарта HTML5, что обеспечивает его переносимость и удобство интеграции в любую современную вычислительную систему.

Для того, чтобы обеспечить бесшовную поддержку интеграции тренажера в качестве серверной службы или веб-приложения, части программного комплекса также разделены на две части: визуальная (front-end) и контролирующая (back-end).

Front-end обеспечивает графическое (или XML – для встраиваемых систем) представление игрового поля и поддержку реализации действий оператора, а также передачу данных об изменении состояния среды в контролирующую часть, которая может быть невидима оператору.

Back-end выполняет подбор тестовых заданий на основе текущего уровня испытуемого, предсказание стратегии агентов, играющих против оператора, а также логирование – фиксацию сообщений о последовательности смены уровней и итоговой оценке оператора – результате работы тренажера. Такие сообщения могут быть переданы в произвольный выходной поток либо подавлены на стороне лица, осуществляющего контроль за процедурой тестирования (рис. 30).

```
Starting server at http://localhost:8080...
Defeat:  [2]
Victory: [2 3]
Defeat:  [2 3 2]
Defeat:  [2 3 2 1]
Victory: [2 3 2 1 2]
```

Рис. 30. Пример сообщений со стороны контролирующей части тренажера.

Для наглядного и детального рассмотрения практического применения тренажера обратимся к схеме данных, отражающей внутреннее взаимодействие компонентов набора программных продуктов, обслуживающих работу многоагентной системы (рис. 31).

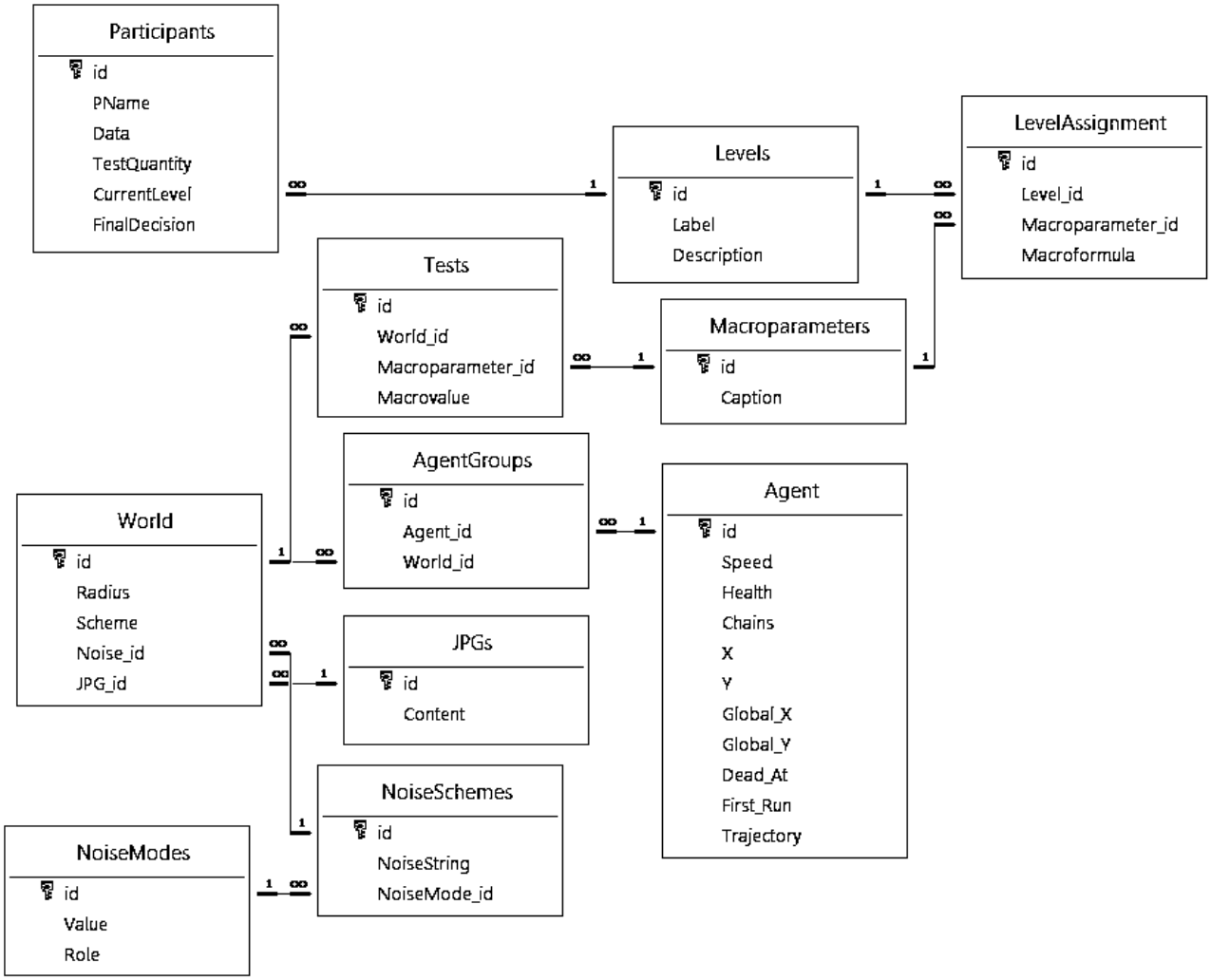

Рис. 31. Схема данных, охватывающая многоагентную модель в целом со всеми программными компонентами (редактор и проигрыватель модели, генератор серии игр для определения макропараметров, тренажер).

Опишем ее содержимое, попутно описывая процессы, сопряженные с заполнением и обработкой данных, начиная от конечного пользователя.

В самом начале пользователь (участник) регистрируется в системе проведения тренировки, и для него определяется имя или псевдоним (в зависимости от поставленных условий тестирования и конкретной реализации программно-вычислительной среды, благодаря которой осуществляется испытание). Его данные, такие, как пароль, а также информация, полученная при регистрации (скан паспорта или других личных документов, согласие на обработку персональных данных и т.д.) хранятся в двоичном зашифрованном виде внутри базы. Испытуемому предварительно задается текущий уровень по умолчанию, обычно – средний из набора существующих (заданных).

Каждая запись в каждой таблице в представленной схеме имеет уникальный первичный ключ-идентификатор (id). Набор участников описывается таблицей Participants. Ее уникальные поля: PName – строка, содержащая псевдоним участника, Data – двоичная информация, описывающая личные данные участника, TestQuantity – количество сеансов, в которых данный пользователь принял участие, FinalDecision – логическое поле, определяющее, была ли в рамках данной сессии определена конечная оценка, соответствующая уровню испытуемого. Поле CurrentLevel – внешний ключ, связанный с идентификатором таблицы Levels. Эта таблица описывает доступные системе тренажера уровни (поле Label определяет метку уровня),

а также их описание (поле Description – например, «новичок» или «средний»). Оба уникальных поля в таблице – текстовые.

Уровни, устанавливающие связь между компетентностью испытуемого и его действиями в игровой ситуации, сопоставляются с макропараметрами, отражающими прогнозируемую выигрышную стратегию. Структура данных сформирована таким образом, чтобы можно было добавлять произвольное количество макропараметров, помимо ранее упомянутых и используемых в данной реализации – это достигается посредством меток – буквенных обозначений конкретных макропараметров с дальнейшей привязкой к конкретному уровню. Поскольку уровню может соответствовать не одно конкретное значение каждого отдельного макропараметра, а диапазон, задается сопровождающая запись, которая воспринимается приложением, извлекающим данные о макропараметрах, как значение или условие в зависимости от типа макропараметра и содержимого формулы, интерпретируемой в контексте рассматриваемого макропараметра (пример значений: «=20», «>5», «50%» и т.д.).

Набор данных, позволяющий определить соответствие между уровнями и макропараметрами, формируется посредством генерации серии игровых ситуаций с заданными начальными условиями. Для каждого из соответствующих этим ситуациям состояний модели определяется значение макропараметров, привязываемых к конкретному уровню. В состояния модели могут быть внесены изменения с помощью внешнего редактора перед экспортом в тренажер. После внесения модели в тренажер она становится доступной испытуемому в виде испытания (при условии, что испытуемому присвоен соответствующий уровень).

Доступные интерпретируемые макропараметры хранятся в таблице Macroparameters с полем Caption, определяющим метку соответствующего

макропараметра. Связь между конкретным уровнем и соответствующим ему набором макропараметров реализуется с помощью таблицы LevelAssignment. Эта таблица содержит два внешних ключа, один для связи с таблицей Levels (Level_id), другой – для связи с таблицей Macroparameters (Macroparameter_id). Таблица дополнена текстовым полем Macroformula, задающим условие установления соответствия между уровнем и значениями каждого из связанных с ним макропараметров.

Набор состояний модели представляется таблицей Tests. Эта таблица связана внешним ключом Macroparameter_id с таблицей Macroparameters, а также внешним ключом World_id с таблицей World, задающей параметры конкретной модели (эта таблица подробно будет рассмотрена далее). Поле, которое хранит конкретное значение макропараметра для определенного набора параметров модели, называется Macrovalue. Значения для этого поля хранятся в текстовом виде, поскольку тип макропараметра, в общем случае, не определен (может быть вещественным, целочисленным и пр.).

Для хранения сгенерированных или созданных вручную игровых полей в программной реализации модели имеется специальная структура, описывающая необходимые данные, которые затем интерпретируются и визуализируются, а также видоизменяются в ответ на реакцию со стороны испытуемого. Эти данные состоят из четырех основных частей: визуальное представление поля (иллюстрация физического объекта), радиус (вещественное число, соответствующее реальным размерам физического объекта), схема вероятностей (совокупность карт осуществимостей и уязвимостей) и модель подавления средств связи между агентами.

Иллюстрация физического объекта представляет собой сжатый рисунок местности (как правило, в качестве такого рисунка выступает снимок со спутника). Схема вероятностей задается нотацией в формате XML,

определяющей размеры радиальной сетки мира (количество кругов C и секторов S), а также набор координат X и Y, задаваемых в пределах этой сетки, причем для каждой координаты задается соответствующее значение (value) карты осуществимости A и уязвимости B. Неоднозначность, появляющаяся как следствие отсутствия указанных значений для какой-либо координаты, решается путем установки значения по умолчанию для соответствующих карт. Пример части XML-разметки, описывающей такую схему, показан на рис. 32.

```
<World C="10" S="10">
    <Coordinate X="3" Y="5">
        <A value="0.005"></A>
        <B value="0.387"></B>
    </Coordinate>


    <!-- ... etc -->

</World>
```

Рис. 32. Пример XML-разметки блока схемы вероятностей.

Модель подавления средств связи реализуется в виде дискретной функции, способной принимать конкретное значение на каждом такте игры. В данной реализации модели подавления диапазон значений предполагает только два допустимых состояния подавления: “0” – отсутствие помех и “1” – блокировка связи. Схема данных допускает дальнейшее расширение набора допустимых состояний (например, «2» – случайное переменное наличие помех), поэтому состояния подавления хранятся в виде их численного

представления и комментария с описанием назначения – роли в игровой ситуации.

Для описания игровых полей в схеме данных представлены четыре таблицы. Таблица World хранит представление игровых моделей с совокупностью начальных параметров и условий. Таблица JPGs содержит иллюстрации моделируемых физических объектов (представленных в двоичном виде в поле Content). Таблица NoiseSchemes хранит набор дискретных функций, описывающих конкретные модели подавления. Таблица NoiseModes определяет доступный диапазон состояний подавления, причем для каждого из состояний определяется его роль (Role, текстовое поле) и значение (Value, текстовое поле – для нативной поддержки шестнадцатеричной нотации).

Таблица World связана внешними ключами с таблицей NoiseSchemes (Noise_id) и JPGs (JPG_id) для задания конкретных значений функции подавления и иллюстрации объекта, соответственно. Вещественное поле Radius определяет значение радиуса объекта, двоичное поле Scheme – сжатое представление разметки XML, описывающей карты осуществимостей и уязвимостей.

Таблица NoiseSchemes содержит внешний ключ NoiseMode_id, связывающий ее с таблицей NoiseModes для определения доступных дискретных значений функций подавления. Текстовое поле NoiseString задает последовательный набор значений конкретной дискретной функции на каждом такте игры от ее начала.

После рендеринга и визуального отображения составляющих игрового поля многоагентной системы, тренажер отслеживает движение агентов согласно алгоритмам, задающим логику их перемещения, а также параметрам, определенным для каждого агента. К таким уникальным (в

общем случае) параметрам относятся такие характеристики, как координаты местоположения, скорость перемещения, момент (номер такта) выхода из строя, траектория движения, степень повреждений (в данной реализации модели нет системы многоуровневого повреждения, хотя схема данных поддерживает ее разработку и внедрение в ходе развития проекта в дальнейшем; поэтому в настоящее время степень повреждений может иметь лишь два значения, указывающих на ситуацию выхода из игры (0) либо активность (1) агента). Все агенты рассматриваются как независимые единицы со своей историей действий, но группируются относительно игровой модели, которой принадлежат, и, соответственно, поля, по которому осуществляют перемещение.

На схеме данных представлены две таблицы, осуществляющие представление агентов как самостоятельных единиц взаимодействия с игровым полем в рамках многоагентной системы. Таблица Agent обеспечивает структуру для задания и хранения характеристик агентов, а таблица AgentGroups обеспечивает связь агентов с игровым полем, в рамках которого для них развивается игровая ситуация. Последнее осуществляется посредством внешних ключей Agent_id (для связи с таблицей Agent) и World_id (для связи с таблицей World).

В таблице Agent определены следующие уникальные поля, хранящие характеристики каждого конкретного агента: Speed (вещественное число, определяющее скорость), X и Y (круг и сектор, соответственно – целочисленные координаты местоположения агента на игровом поле), Global_X и Global_Y (вещественные координаты местоположения агента в масштабе реального географического объекта, соответствующего игровому полю), Health (целое число, степень повреждений), Dead_At (целое число,

хранящее положительное значение, соответствующее номеру такта, на котором агент вышел из строя, либо отрицательное, пока объект жив).

Для хранения истории действий, а также промежуточного состояния работы алгоритмов модели, отвечающих за выбор направления движения исходя из текущей ситуации, таблица дополнена тремя особыми полями. Поле Chains – двоичное, хранит закодированную информацию о предпочтениях агента относительно выбора направления движения по игровому полю на текущем такте игры. Поле First_Run – логическое, определяет, осуществлял ли агент движение по полю ранее (эти сведения необходимы для обеспечения более ресурсоемкой процедуры оптимизации). Поле Trajectory – двоичное, хранит в зашифрованном виде историю совершенных агентом перемещений в системе координат реального географического объекта, моделируемого в ходе игры).

Система тренажера отбирает агентов, соответствующих набору параметров конкретной модели. Эта модель, в свою очередь, обусловлена макропараметрами, определенными для текущего уровня испытуемого в рамках данной сессии. Взаимодействие с этими агентами через графический интерфейс пользователя, предоставляемый программной реализацией тренажера, изменяет состояние агентов и статус текущей игры, в зависимости от которого меняется или устанавливается уровень испытуемого. Если уровень испытуемого полагается установленным, то соответствующий флаг о принятии окончательного решения переключается, а процесс тестирования завершается; в противном случае генерируется новая игровая ситуация (с увеличением счетчика количества испытаний для текущего испытуемого), и процесс отбора данных по схеме начинается снова, исходя из нового уровня испытуемого и текущих параметров сессии.

Исследование взаимодействия частей программного комплекса тренажера с пользователем-оператором в нотации описания бизнес-процессов

Разобрав структуру данных и алгоритмическое представление проекта, рассмотрим тренажер с точки зрения бизнес-процесса в контексте взаимодействия испытуемого с комплексом программ, обеспечивающих определение уровня навыков, а также его формирование у пользователя тренажера (рис. 33).

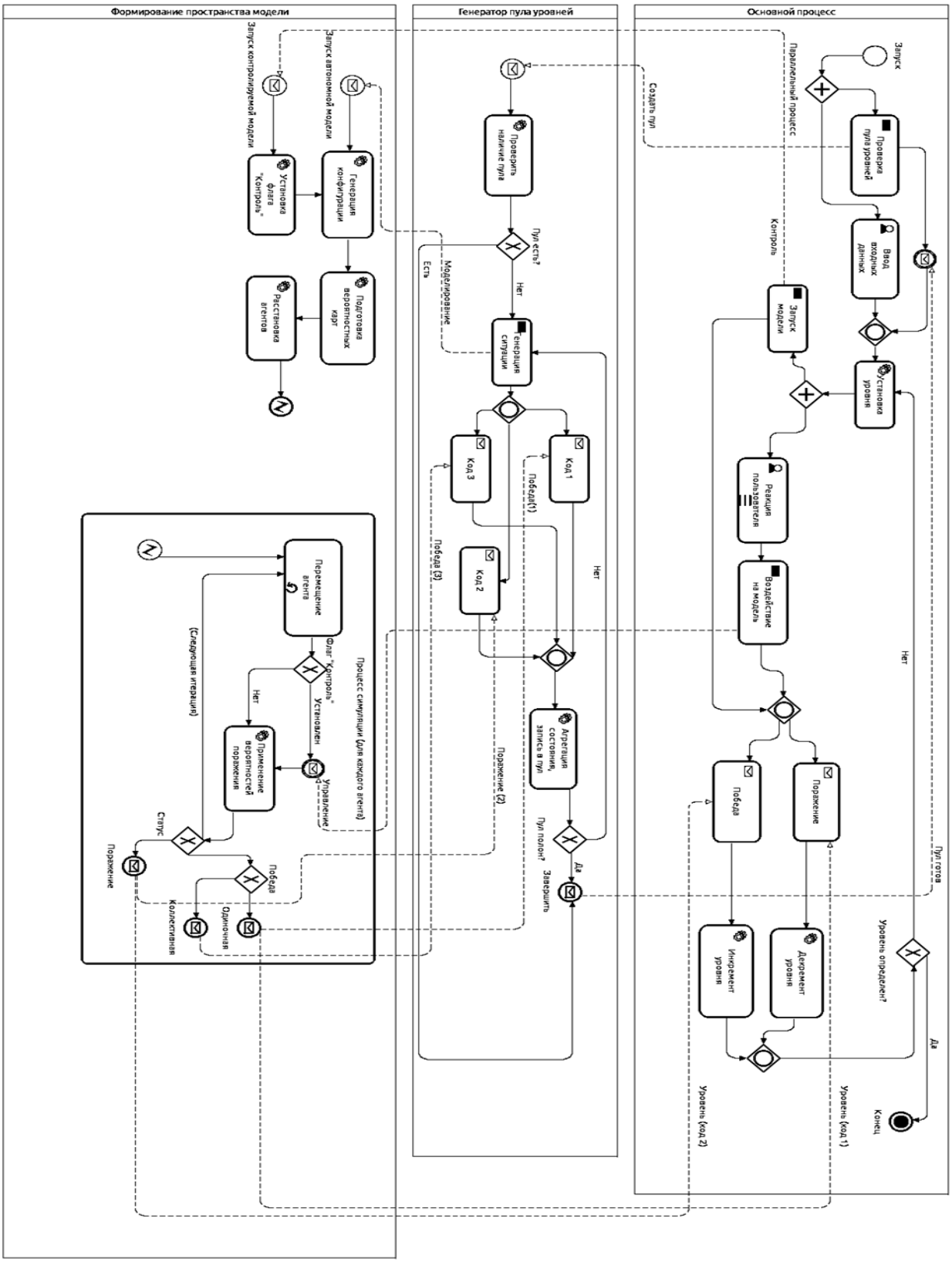

Рис. 33. Схема программного комплекса в нотации BPM.

Для наглядного отображения процесса формирования навыков и определения уровня в ходе использования тренажера построим схемы в нотации BPM [105] для описания каждого процесса, с тем чтобы последовательно и детально рассмотреть происходящие процессы, место пользователя в их работе, а также основные программные процессы, задействованные в ходе выполнения вычислительных операций для связи модели, механизма вычисления макропараметров и их применения в контексте виртуального окружения, с которым взаимодействует испытуемый посредством тренажера.

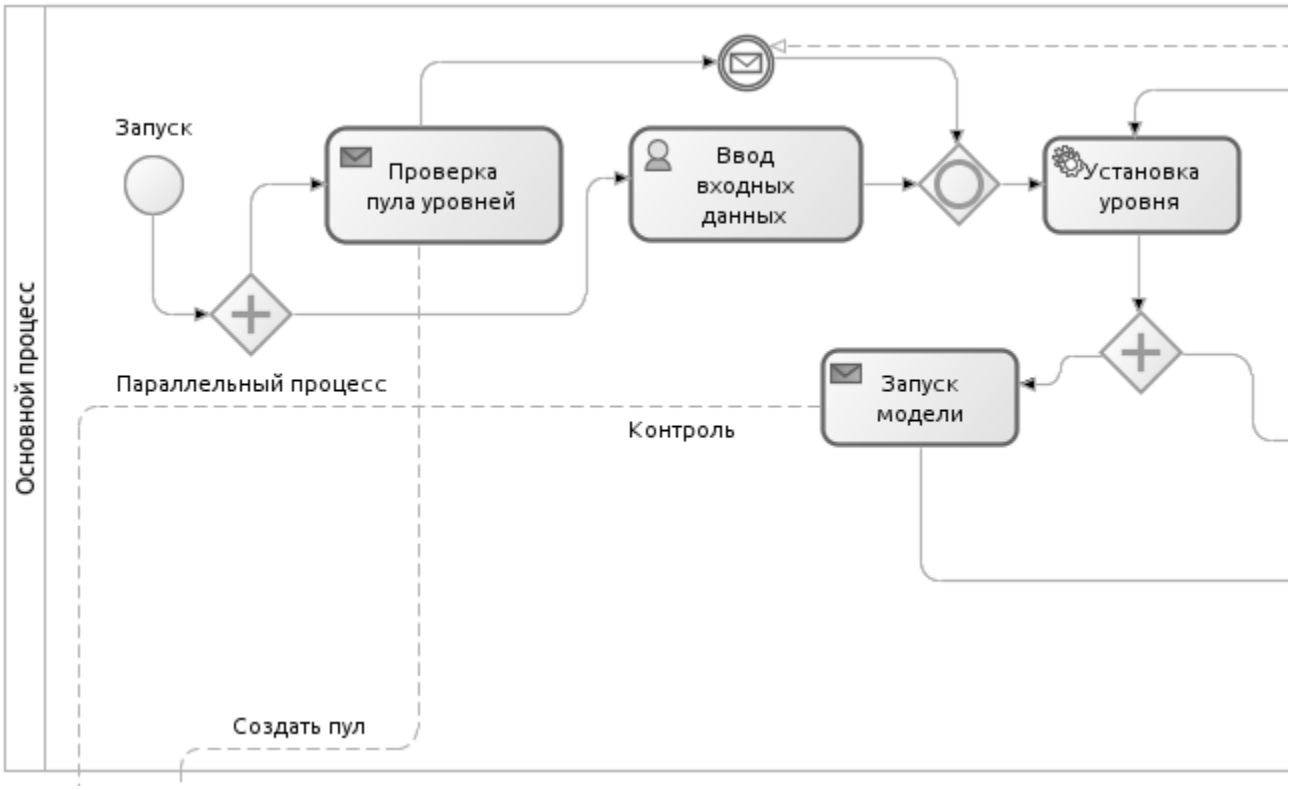

Рис 34. Основной процесс (начало).

Программа, обеспечивающая работу тренажера как целостного комплекса, представляет собой кросс-платформенное приложение с графическим кросс-браузерным интерфейсом на базе языка разметки гипертекста с поддержкой стандарта HTML5, что обеспечивает его

переносимость и удобство интеграции в любую современную вычислительную систему. Пользователь запускает основной процесс тренажера в момент входа в соответствующий веб-интерфейс с указанием личных данных (рис. 34). Как абстракция, основной процесс отвечает за корректное взаимодействие компонентов тренажера, обработку вводимых данных (со стороны пользователя-испытуемого) и наглядное, с точки зрения лица, контролирующего процесс обучения (системного администратора), представление достигнутого пользователем уровня. Этот уровень представляется в численном виде и интерпретируется лицом, контролирующим процесс обучения, в зависимости от рода поставленных задач.

Запуск тренажера параллельно активирует пул уровней, собираемый фоновым процессом на основе запуска многочисленных моделей с различными значениями макропараметров, соответствующих определенному ранее в базе данных диапазону уровней. Предполагается, что этот процесс требуется проводить только при первом запуске либо в ходе обновления системы тренажера.

Если пул сгенерирован, а пользователь ввел свои личные данные, запускаются два параллельных процесса, один из которых – фоновый, для запуска модели с параметрами, соответствующими текущему уровню испытуемого. Второй процесс, также фоновый, ожидает реакции пользователя на события модели для оказания воздействия на ход моделируемой ситуации. Активный процесс ожидает завершения работы модели с тем или иным исходом – поражением испытуемого либо его победой (рис. 35).

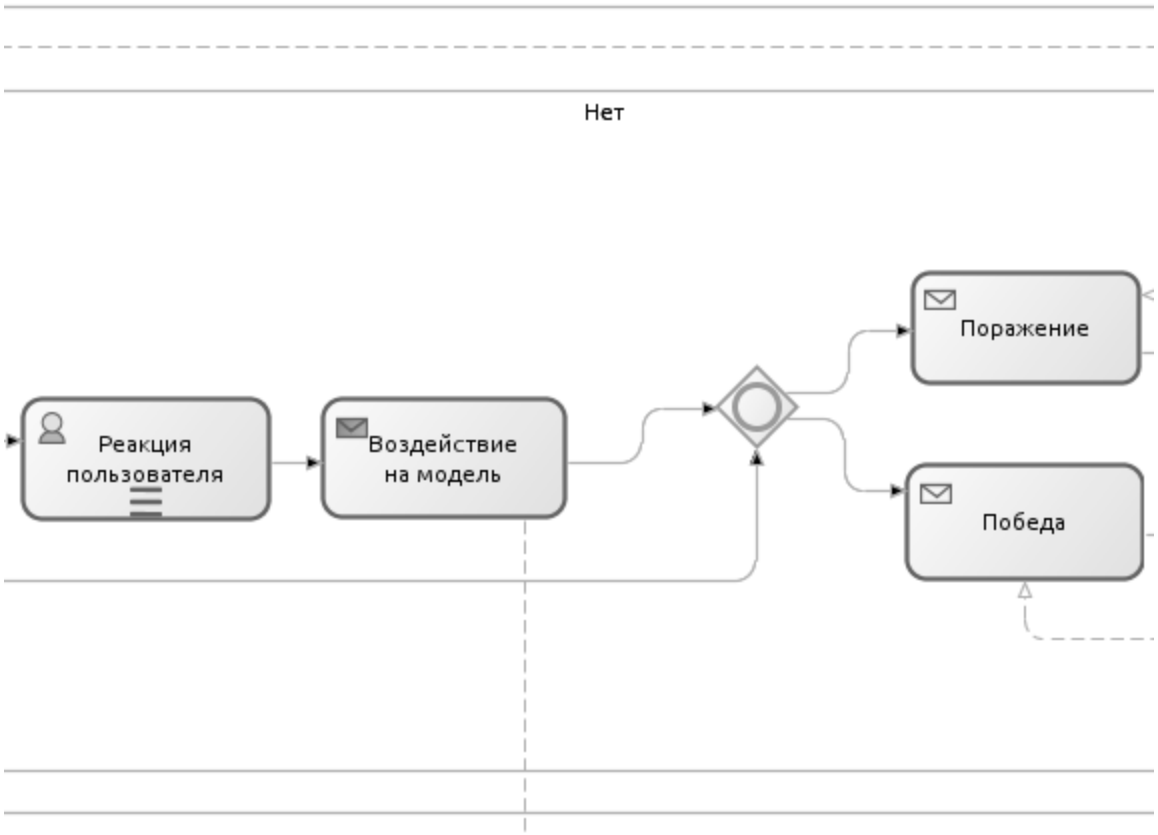

Рис 35. Основной процесс (продолжение).

Воздействие на модель интерпретируется как отправка сообщения, которое, согласно внутреннему интерфейсу библиотеки, отвечающей за генерацию моделируемой ситуации, необходимо перехватить и интерпретировать в соответствии с доступным для испытуемого набором манипуляторов (из числа обрабатываемых моделью). В самом общем случае это может быть клавиатура и мышь, но на специализированных тренажерных комплексах, где код тренажера выполнен как часть встроенной системы, это может быть пульт, набор рычагов или приборная панель, в зависимости от имитируемого устройства, для которого осуществляется процедура тренировки.

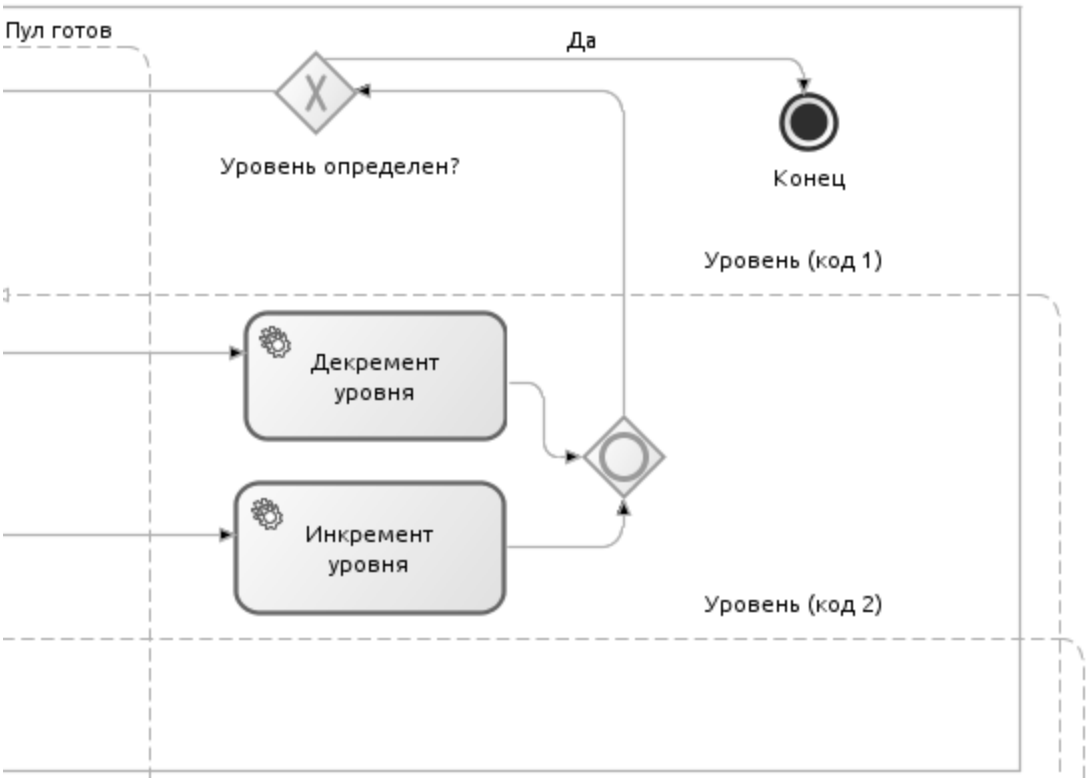

Рис 36. Основной процесс (завершение).

Уровень испытуемого изменяется согласно результату моделирования; если оговоренные ранее условия для определения текущего уровня испытуемого выполнены, основной процесс завершается.

Сигнал о проверке пула уровней принимает специальный процесс (рис. 37). Этот процесс отвечает за генерацию пула уровней в случае его отсутствия.

Генерация осуществляется путем создания ситуаций с учетом разнообразных макропараметров и автоматических экспериментов, а также корректирующей информации со стороны лица, ответственного за разработку тестовых заданий.

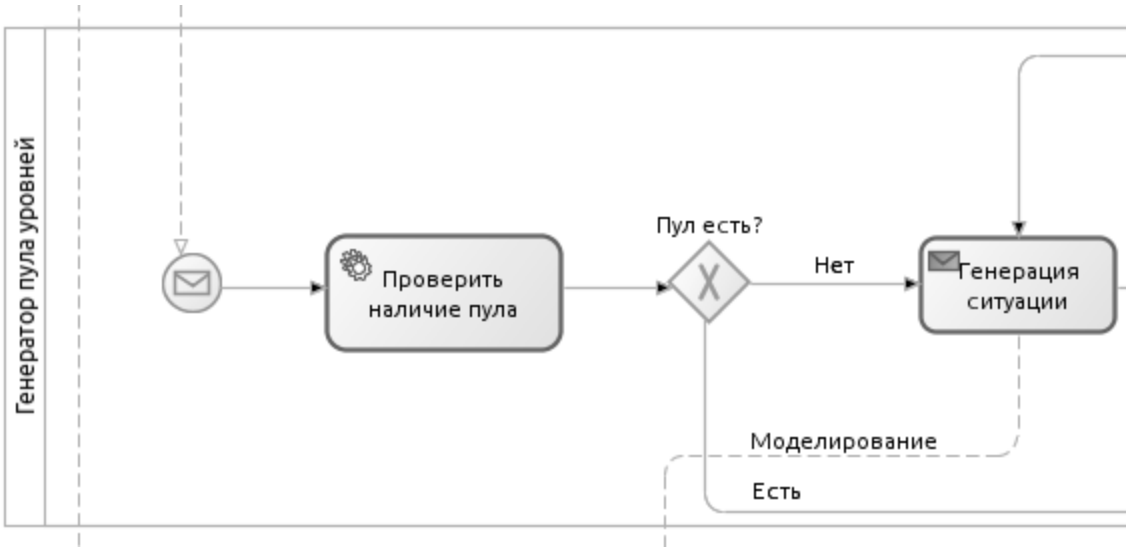

Рис 37. Генератор пула уровней (начало).

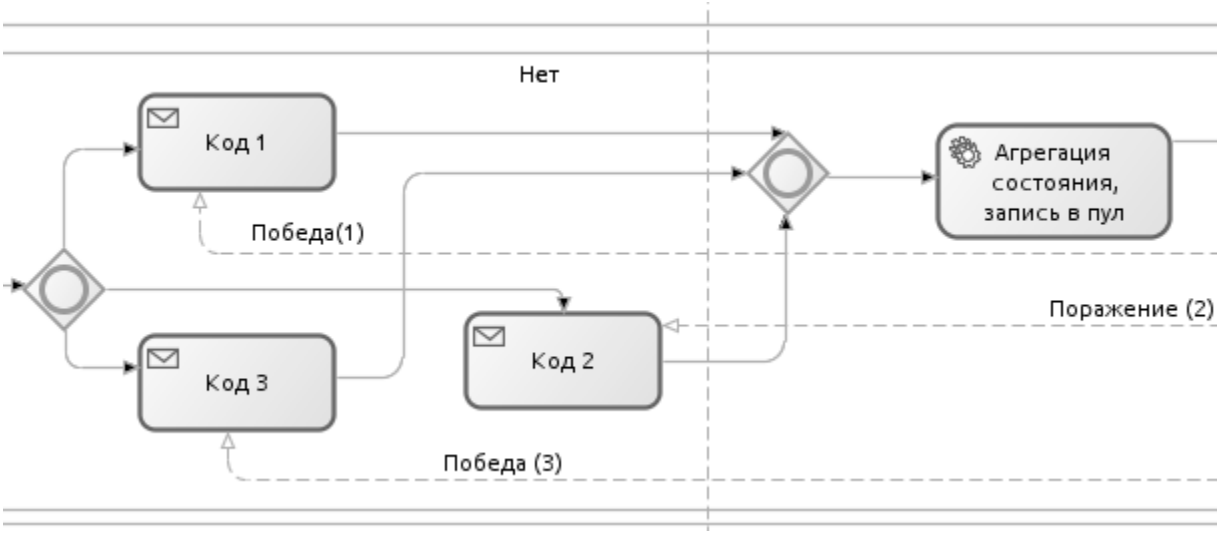

Рис 38. Генератор пула уровней (продолжение).

Для каждого состояния, задающего уровень, неоднократно запускается модель, а код возвращаемого результата, определяющий поражение либо победу, интерпретируется в соответствии с марковской моделью, отвечающей за идентификацию макропараметров и интенсивность переходов между состояниями (рис. 38). Процесс завершается, если пул заполнен, то есть хранит в себе оговоренный объем проведенных экспериментов (рис. 39).

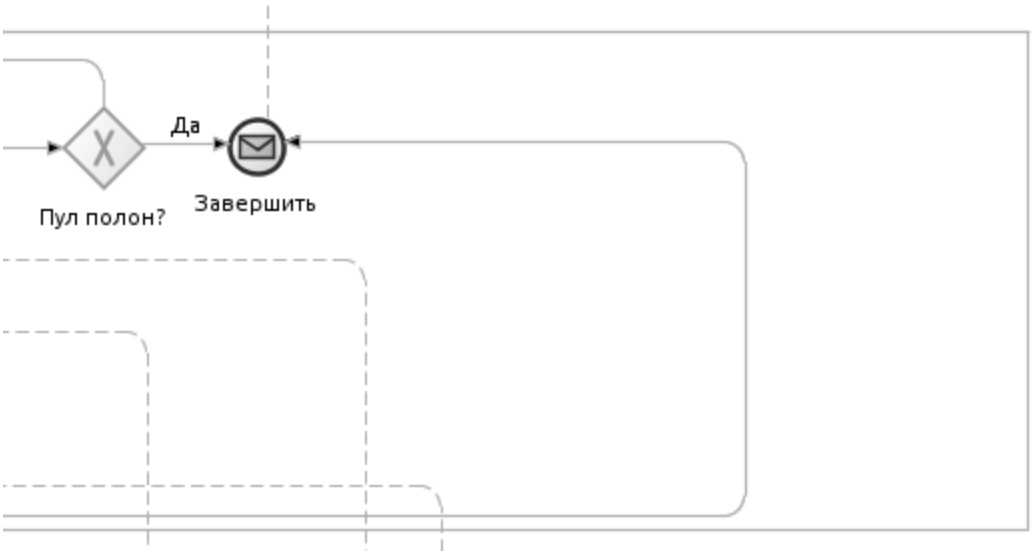

Рис 39. Генератор пула уровней (окончание).

Основным низкоуровневым, по сравнению с другими составными частями комплекса тренажера, модулем, обеспечивающим корректную работу описанных выше процессов, является библиотека, отвечающая за генерацию моделируемых ситуаций на основе переданных параметров и реализующая процесс, отвечающий за формирование пространства модели (рис. 40).

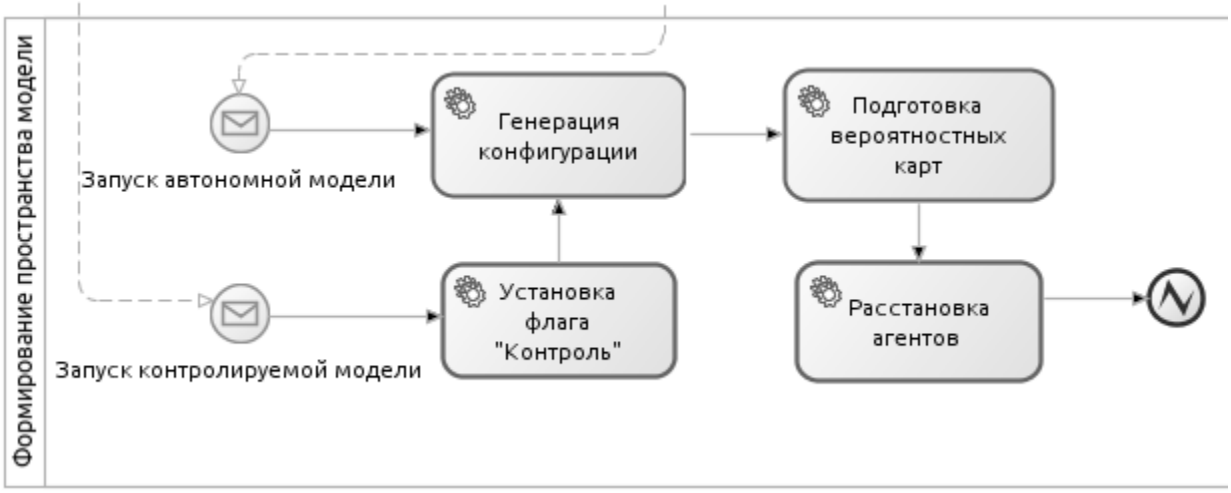

Рис 40. Формирование пространства модели (начало).

В зависимости от процесса, который послал сигнал на генерацию новой модели, процесс, отвечающий за формирование пространства модели, переходит в автономный либо контролируемый режим.

В автономном режиме предполагается, что участие пользователя отсутствует, а течение моделируемой ситуации характеризуется исключительно алгоритмом, задающим изменение модели при последовательном переходе между тактами.

В контролируемом режиме ожидается воздействие пользователя на моделируемую ситуацию посредством передаваемых входных (по отношению к библиотеке) данных. В этом режиме устанавливается глобальный условный флаг «Контроль», обеспечивающий перехват и обработку действий пользователя.

Вне зависимости от выбранного режима, перед запуском модели выполняется подготовка: передаваемые параметры преобразуются в такой вид, чтобы по ним можно было построить общую конфигурацию и набор данных для соответствующей модели (количество кругов, секторов, участвующих в модели агентов, физическая модель для корректного перевода координат, дискретная функция помех и другая техническая информация). Затем осуществляется заполнение модели: построение карт осуществимостей и уязвимостей, а также расстановка и инициализация агентов на игровом поле в заданных координатах.

Когда модель готова к запуску, по виртуальному прерыванию на каждом такте для каждого агента на поле вызывается подчиненный процесс-симулятор, обеспечивающий взаимодействие агентов, цели и моделируемого пространства (рис. 41).

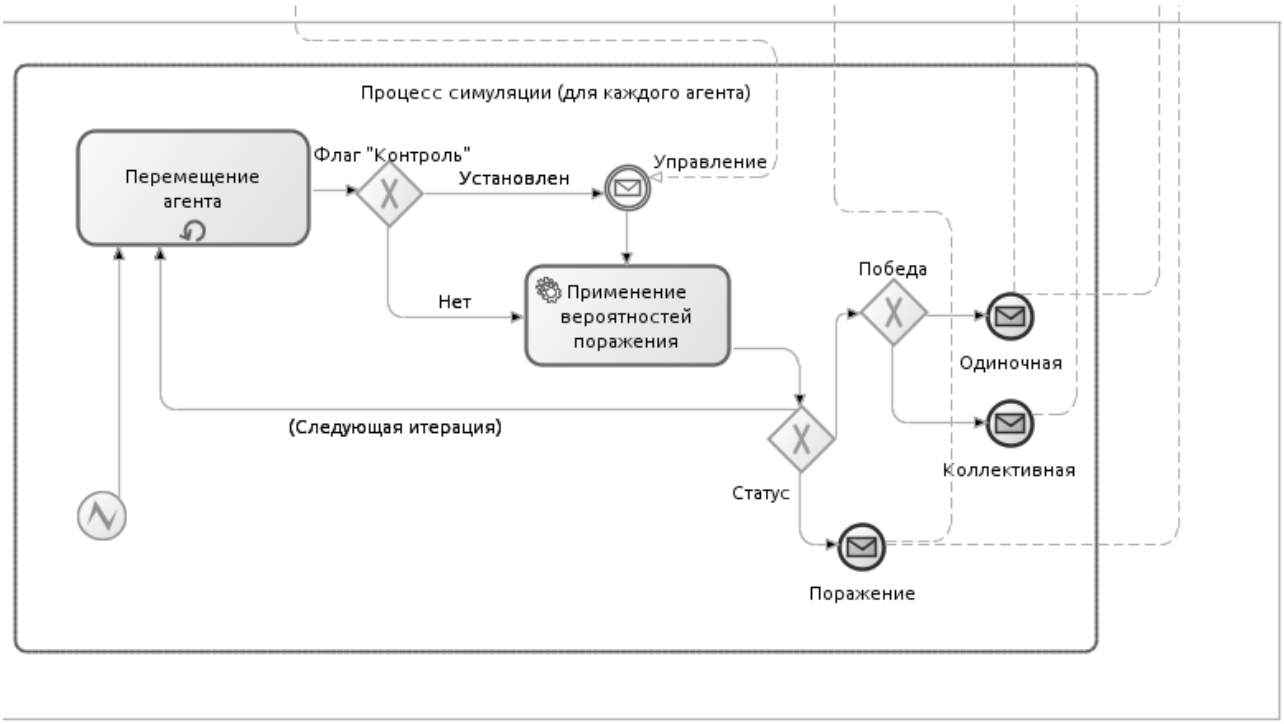

Рис 41. Формирование пространства модели (окончание).

Процесс на каждой итерации осуществляет движение агента и проверку наличия контроля со стороны пользователя; в случае, если контроль установлен, действия пользователя считываются, интерпретируются и учитываются при применении вычисляемых вероятностей поражения агентов и цели. Если вычисления показывают, что статус моделируемой ситуации изменился на один из возможных конечных («Одиночная победа», «Коллективная победа» или «Поражение»), то процесс останавливается и возвращает вызвавшему его процессу соответствующий код статуса завершения ситуации. В противном случае происходит переход к следующей итерации. Модель, таким образом, работает до тех пор, пока уровень компетентности испытуемого (оператора) не будет определен средствами комплекса программ «Тренажер» согласно описанным критериям.

Тренажер позволяет избежать затрат, связанных с эксплуатацией и сопутствующим износом или выходом из строя реального дорогостоящего оборудования. Тренажер обеспечивает механизм по контролю за уровнем компетентности оператора в реальном времени с помощью макропараметров. Реализованная комплексная схема данных поддерживает последующее расширение текущих возможностей модели через добавление новых (точная система повреждений агента, более гибкая модель подавления сигнала и т.п.). Программная реализация тренажера, представленная на рисунке 42, поддерживает кросс-платформенность и переносимость, а также может быть использована в качестве сервера для поддержки веб-приложения или сервиса.

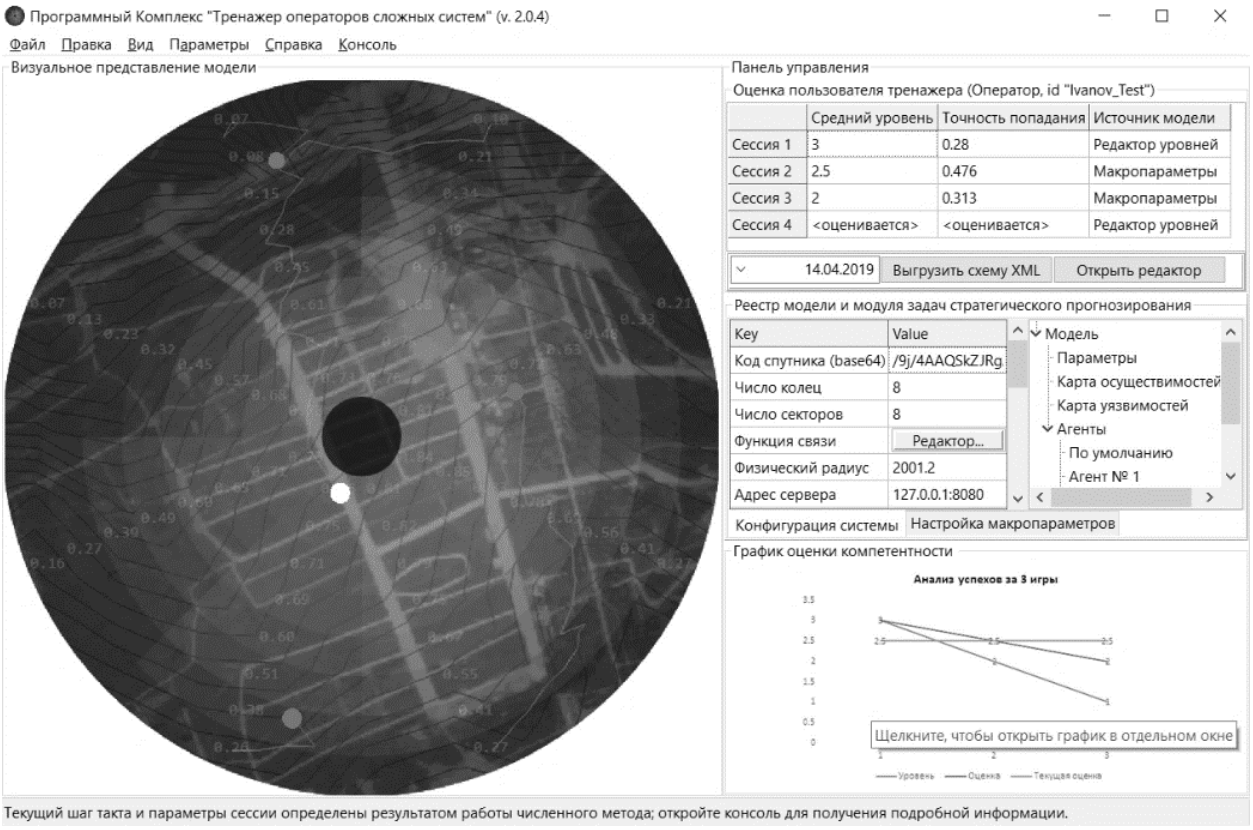

Рисунок 42. Интерфейс программного комплекса.

Заключение

Основные результаты, выносимые на защиту

1. Математическая модель и алгоритм поведения прикладной многоагентной системы. Модель представлена марковским процессом с дискретными состояниями и непрерывным временем и обеспечивает коллективное и автономное поведение агентов, а также их недетерминированное перемещение по рассматриваемой области поверхности. К особенностям модели и ее программной реализации относятся:
   – способность обучаться без сохранения предыстории действий;
   – способность быстро приспосабливаться к изменениям значений параметров, характеризующих свойства моделируемой системы;
   – кроссплатформенность, переносимость и расширяемость программной реализации;
   – способность модели удовлетворять представленным требованиям на программно-аппаратных вычислительных системах с ограниченным количеством ресурсов без принципиальных потерь в скорости принятия решений.
2. Математическая модель и метод прогнозирования, обеспечивающие оперативную оценку ресурсов, необходимых для решения задачи, на основе количественных критериев. Особенностями модели являются:
   – применение эмпирически подобранных макропараметров, определяющих состояние системы и качественные

характеристики моделируемой ситуации на заданном такте времени;

- комплекс программ, реализующих модель, осуществляющих ансамбль экспериментальных вычислений в автоматическом режиме и позволяющих наглядно представить конечную информацию стратегического характера для лица, принимающего решение.

3. Адаптивный численный метод оптимизации, используемый при идентификации параметров прикладной многоагентной системы. Эффективность метода подтверждена серией вычислительных экспериментов.
4. Комплексы программ для прогнозирования поведения прикладной многоагентной системы и оценки уровня подготовки и адаптивного обучения операторов, работающих с этой системой.
5. Аналитические выражения для вероятности поражения цели агентом в случае его простейшего поведения.

Список литературы

1. Городецкий В.И., Карсаев О.В., Самойлов В.В., Серебряков С.В. Прикладные многоагентные системы группового управления. – Искусственный интеллект и принятие решений, №2, 2009, с. 3–24.
2. Осипов Г.С. Методы искусственного интеллекта. – М.: Физматлит, 2011. – 296 с.
3. Фон Нейман Дж. Теория самовоспроизводящихся автоматов. – М.: URSS, 2010. – 384 с.
4. Цетлин М.Л. Исследования по теории автоматов и моделированию биологических систем. – М: Наука, 1969. – 316 с.
5. Куравский Л.С., Юрьев Г.А. Адаптивное тестирование как марковский процесс: модели и их идентификация. - Нейрокомпьютеры: разработка и применение, №2, 2011, с. 21-29.
6. Марковские модели в задачах диагностики и прогнозирования: Учеб. пособие. / Под ред. Л.С. Куравского. – М.: РУСАВИА, 2013. – 172 с.
7. Kuravsky L.S., Marmalyuk P.A., Baranov S.N., Alkhimov V.I., Yuryev G.A. and Artyukhina S.V. A New Technique for Testing Professional Skills and Competencies and Examples of its Practical Applications. – Applied Mathematical Sciences, Vol. 9, 2015, no. 21, 1003–1026, http://dx.doi.org/10.12988/ams.2015.411899.
8. Kuravsky L.S., Marmalyuk P.A., Yuryev G.A. and Dumin P.N. A Numerical Technique for the Identification of Discrete-State Continuous-Time Markov Models. – Applied Mathematical Sciences, Vol. 9, 2015, no. 8, 379–391, http://dx.doi.org/10.12988/ams.2015.410882.
9. Kuravsky L.S., Marmalyuk P.A., Yuryev G.A., Belyaeva O.B. and Prokopieva O.Yu. Mathematical Foundations of Flight Crew Diagnostics

Based on Videooculography Data. – Applied Mathematical Sciences, Vol. 10, 2016, no. 30, 1449–1466, http://dx.doi.org/10.12988/ams.2016.6122.

10. Aras R., Dutech A., Charpillet F. Cooperation through communication in decentralized Markov games. In International Conference on Advances in Intelligent Systems - Theory and Applications - AISTA’2004, Luxembourg-Kirchberg/Luxembourg. 2004.
11. Boutilier C. Planning, learning and coordination in multiagent decision processes. In Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge, Morgan Kaufmann Publishers Inc. 1996. - pp. 195–210.
12. Claus C. & Boutilier C. The dynamics of reinforcement learning in cooperative multiagent systems. In Proceedings of the National Conference on Artificial Intelligence, John Wiley & Sons Ltd. 1998. - pp. 746–752.
13. Miikkulainen R. Creating Intelligent Agents in Games (2006). The Bridge: 5-13, 2006.
14. Owen G. Game Theory. Academic Press. 1995.
15. Read M., Möslinger Ch., Dipper T., Kengyel D., Hilder J., Thenius R., Tyrrell A., Timmis J., Schmickl T. Profiling Underwater Swarm Robotic Shoaling Performance using Simulation. In Proceedings of TAROS 2013 (2013), 456-462.
16. Snodgrass S. and Ontanon S. A hierarchical mdmc approach to 2d video game map generation. In Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2015.
17. Куравский Л.С., Марголис А.А., Юрьев Г.А., Мармалюк П.А. Концепция системы поддержки принятия решений для психологического тестирования // Психологическая наука и образование, 2012, №1. с.56–65.

18. Куравский Л.С., Мармалюк П.А., Алхимов В.И., Юрьев Г.А. Математические основы нового подхода к построению процедур тестирования // Экспериментальная психология, 2012, т. 5, №4, с. 75–98.
19. Куравский Л.С., Марголис А.А., Мармалюк П.А., Юрьев Г.А., Думин П.Н. Обучаемые марковские модели в задачах оптимизации порядка предъявления психологических тестов // Нейрокомпьютеры: разработка и применение, 2013, №4, с. 28–38.
20. Куравский Л.С., Мармалюк П.А., Алхимов В.И., Юрьев Г.А. Новый подход к построению интеллектуальных и компетентностных тестов // Моделирование и анализ данных, 2013, №1, с. 4-28.
21. Куравский Л.С., Мармалюк П.А., Барабанщиков В.А., Безруких М.М., Демидов А.А., Иванов В.В., Юрьев Г.А. Оценка степени сформированности навыков и компетенций на основе вероятностных распределений глазодвигательной активности // Вопросы психологии, 2013, №5, с. 64–81.
22. Попков С.И., Куравский Л.С. Свидетельство о государственной регистрации программы для ЭВМ №2017618950 "Программа для моделирования стохастического поведения прикладной многоагентной системы (St#MAS)" /Правообладатели Попков С.И., Куравский Л.С. (Россия). — Заявка №2017615896; Заяв. 20.06.2017; Зарегистр. 11.08.2017.—(РОСПАТЕНТ).
23. Антонио Джулли, Суджит Пал. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow / пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2018. – 294 с.: ил.

24. Люгер, Джордж, Ф. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 864 с.: ил. – Парал. тит. англ.
25. Шакла Нишант. Машинное обучение и TensorFlow. – СПб.: Питер, 2019. – 336 с.: ил. – (Серия «Библиотека программиста»).
26. Рассел Стюарт, Норвиг Питер. Искусственный интеллект: современный подход, 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2018. – 1408 с.: ил. – Парал. тит. англ.
27. Гудфеллоу Я., Бенджио И. Курвиль А. Глубокое обучение / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2017. – 652 с.: ил.
28. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018. – 480 с.: ил. – (Серия «Библиотека программиста»).
29. Паттерсон Дж., Гибсон А. Глубокое обучение с точки зрения практика / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2018 – 418 с.: ил.
30. Шолле Франсуа. Глубокое обучение на Python. – СПб.: Питер, 2018. – 400 с.: ил. – (Серия «Библиотека программиста»).
31. Траск Эндрю. Грокаем глубокое обучение. – СПб.: Питер, 2019. – 352 с.: ил. – (Серия «Библиотека программиста»).
32. Равичандиран Судхарсан. Глубокое обучение с подкреплением на Python. OpenAI Gym и TensorFlow для профи. – СПб.: Питер, 2020. – 320 с.: ил. – (Серия «Библиотека программиста»).
33. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. 3-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004.
34. Margaret A. Boden. AI: Its Nature and Future. – Oxford University Press, 2016.
35. Осипов Г.С. Динамические интеллектуальные системы. – Искусственный интеллект и принятие решений, № 1, 2008, с. 47–54.

36. Куравский Л.С., Попков С.И. Вероятностная модель поведения прикладной многоагентной системы. – Нейрокомпьютеры: разработка, применение, № 9, 2016, с. 22-34.
37. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечёткие системы. – М.: Горячая линия-Телеком, 2013. – 384 с.
38. Попков С.И. Программная реализация вероятностной модели поведения прикладной многоагентной системы. – Нейрокомпьютеры: разработка, применение, № 9, 2016, с. 35-44.
39. Попков С.И. Программная реализация вероятностной модели поведения прикладной многоагентной системы (тезисы). XV Всероссийская научная конференция «Нейрокомпьютеры и их применение». МГППУ, 2017. – 2 с.
40. Флэнаган Дэвид. JavaScript: карманный справочник, 3-е изд. : Пер. с англ. – М.: ООО "И.Д. Вильямс", 2015. – 320 с. : ил. – Парал. тит. англ.
41. Керниган Б., Ритчи Д. Язык Программирования Си. Пер. с англ., 3-е изд., испр. – СПб.: "Невский диалект", 2001. – 352 с.: ил.
42. Шилдт Герберт. Java. Полное руководство. 8-е изд.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2013.
43. Earle Castledine. Jump Start CoffeeScript - SitePoint Pty. Ltd., 2012.
44. Interprocess Communication and Networking — URL: http://docs.python.org/3/library/ipc.html (дата обращения 31.01.2018 г.)
45. How to create windows golang DLL and load into C, or delphi, or freepascal — URL: http://github.com/z505/goDLL (дата обращения 31.01.2018 г.)
46. Neil Mitchell's Haskell Blog. Haskell DLL's on Windows — URL: http://neilmitchell.blogspot.ru/2009/11/haskell-dlls-on-windows.html (дата обращения 31.01.2018 г.)

47. Windows for Business — URL: http://www.microsoft.com/en-us/windowsforbusiness (дата обращения 31.01.2018 г.)
48. Build your future with Windows Server — URL: http://www.microsoft.com/en-us/cloud-platform/windows-server (дата обращения 31.01.2018 г.)
49. Issue: cmd/go: -buildmode=c-shared should work on windows — URL: http://github.com/golang/go/issues/11058 (дата обращения 31.01.2018 г.)
50. Issue: Go + Windows != DLL — URL: http://github.com/golang/go/issues/15301 (дата обращения 31.01.2018 г.)
51. Issue: go1.10 build c-shared for to windows dll — URL: http://github.com/golang/go/issues/23052 (дата обращения 31.01.2018 г.)
52. Functional Programming vs. Imperative Programming (C#) — URL: http://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/functional-programming-vs-imperative-programming (дата обращения 31.01.2018 г.)
53. Зубков С. В. Assembler. Для DOS, Windows и UNIX. – М.: ДМК Пресс, 2012.
54. Concurrent Execution — URL: http://docs.python.org/3/library/concurrency.html (дата обращения 31.01.2018 г.)
55. Synchronization primitives — URL: http://docs.python.org/3/library/asyncio-sync.html (дата обращения 31.01.2018 г.)
56. TIOBE Index - URL: http://www.tiobe.com/tiobe-index/ (дата обращения 31.01.2018 г.)
57. Аблязов Р. З. Программирование на Ассемблере на платформе x86-64. – М.: ДМК Пресс, 2016. – 302 с.: ил.

58. Introduction to Access SQL — URL: http://support.office.com/en-us/article/Introduction-to-Access-SQL-d5f21d10-cd73-4507-925e-bb26e377fe7e (дата обращения 31.01.2018 г.)
59. Bryan O'Sullivan, John Goerzen, Don Stewart. Real World Haskell. – O'Reilly Media Inc., 2009.
60. Душкин Р. В. Функциональное программирование на языке Haskell. – М.: ДМК Пресс, 2007.
61. Kotlin Language Documentation — URL: http://kotlinlang.org/docs/kotlin-docs.pdf (дата обращения 31.01.2018 г.)
62. CoffeeScript — URL: http://coffeescript.org (дата обращения 31.01.2018 г.)
63. Babel is a JavaScript compiler — URL: http://babeljs.io (дата обращения 31.01.2018 г.)
64. Lazarus – URL: http://www.lazarus-ide.org/ (дата обращения 31.01.2018 г.)
65. Грас Дж. Data Science. Наука о данных с нуля: Пер. с англ. – СПб.: БХВ-Петербург, 2018. – 336 с.: ил.
66. Донован Алан А. А., Керниган Брайан У. Язык программирования Go. : Пер. с англ. – М.: ООО "И.Д. Вильямс", 2016. – 432 с.: ил. – Парал. тит. англ.
67. Issue: runtime: support dlclose with -buildmode=c-shared — URL: http://github.com/golang/go/issues/11100 (дата обращения 31.01.2018 г.)
68. ctypes — A foreign function library for Python — URL: http://docs.python.org/3/library/ctypes.html (дата обращения 31.01.2018 г.)
69. Душкин Р. В. Практика работы на языке Haskell. – М.: ДМК Пресс, 2010. – 288 с., ил.

70. Hackage :: [Package] — URL: http://hackage.haskell.org (дата обращения 31.01.2018 г.)
71. Kotlin Blog — URL: http://blog.jetbrains.com/kotlin/ (дата обращения 31.01.2018 г.)
72. JetBrains Company Blog — URL: http://blog.jetbrains.com/blog/2013/12/16/jetbrains-st-petersburg-rd-lab-grows-moves-to-new-office/ (дата обращения 31.01.2018 г.)
73. Get Started with Kotlin on Android — URL: http://developer.android.com/kotlin/get-started.html (дата обращения 31.01.2018 г.)
74. Kotlin/Native — URL: http://kotlinlang.org/docs/reference/native-overview.html (дата обращения 31.01.2018 г.)
75. Kotlin/Native infrastructure — URL: http://github.com/JetBrains/kotlin-native (дата обращения 31.01.2018 г.)
76. Samples — URL: http://github.com/JetBrains/kotlin-native/tree/master/samples (дата обращения 31.01.2018 г.)
77. Arrays and pointers — URL: http://docs.python.org/3/library/ctypes.html#arrays-and-pointers (дата обращения 31.01.2018 г.)
78. Шлее М. Qt 5.3. Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2015. – 928 с.: ил. – (В подлиннике)
79. Цуканова Н. И., Дмитриева Т. А. Логическое программирование на языке Visual Prolog. Учебное пособие для ВУЗов. – М.: Горячая линия – Телеком, 2008. – 144 с.: ил.
80. Эккель Б. Философия Java. Библиотека программиста. 4-е изд. – СПб.: Питер, 2013. – 640 с.: ил. – (Серия «Библиотека программиста»).

81. Фултон Х. Программирование на языке Ruby. – М.: ДМК Пресс, 2007. – 688 с.: ил.
82. Тейт Брюс. Семь языков за семь недель. Практическое руководство по изучению языков программирования / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2014. – 384 с.: ил.
83. Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. Классика Computer Science. 4-е изд. – СПб.: Питер, 2012. – 784 с.: ил.
84. Жемеров Д., Исакова С. Kotlin в действии. / пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2018. – 402 с.: ил.
85. Марков В. Н. Современное логическое программирование на языке Visual Prolog 7.5: учебник. – СПб.: БХВ-Петербург, 2016. – 544 с.: ил. – (Учебная литература для ВУЗов).
86. Ben Klemens. 21st Century C, Second Edition. – O'Reilly Media Inc., 2015.
87. HTML 5.2. W3C Recommendation, 14 December 2017 – URL: https://www.w3.org/TR/html5/ (дата обращения: 30.01.2019 г.)
88. Овчаров Л. А. Прикладные задачи теории массового обслуживания. – М.: Машиностроение, 1969. – 324 с.
89. Куравский Л.С., Мармалюк П.А., Юрьев Г.А., Думин П.А. Численные методы идентификации марковских процессов с дискретными состояниями и непрерывным временем. – Математическое Моделирование, № 5, 2017, С. 133-146.
90. Kuravsky L. S., Yuryev G. A., Adaptive testing as a Markov process: Models and their identification, Neurocomputers: Development, Application 2:21–29, 2011.

91. Куравский Л.С., Попков С.И. Представление общих закономерностей поведения многоагентной системы с помощью её макропараметров. — Нейрокомпьютеры: разработка, применение, № 1, 2018, с. 13-25.
92. Попков С.И. Метод внешней оптимизации для идентификации марковских процессов. – Информационные Технологии, № 10, 2018, с. 633-641.
93. Попков С.И. Программная реализация межъязыкового взаимодействия на базе динамических библиотек. – Нейрокомпьютеры: разработка, применение, № 3, 2018, с. 39-49.
94. L. Kuravsky, G. Yuryev. On the approaches to assessing the skills of operators of complex technical systems - In: Proc. 15th International Conference on Condition Monitoring & Machinery Failure Prevention Technologies, Nottingham, UK, September 2018. - 1 pp.
95. Куравский Л. С., Марголис А. А., Мармалюк П. А., Панфилова А. С., Юрьев Г. А. Математические аспекты концепции адаптивного тренажера // Психологическая наука и образование. 2016. Т. 21. No 2. С. 84–95. doi: 10.17759/pse.2016210210
96. L.S. Kuravsky, A.A. Margolis, P.A. Marmalyuk, A.S. Panfilova, G.A. Yuryev, P.N. Dumin. A probabilistic model of adaptive training – Applied Mathematical Sciences, Vol. 10, 2016, no. 48, 2369-2380 – URL: http://dx.doi.org/10.12988/ams.2016.65168 (дата обращения: 30.01.2019 г.)
97. Тренажерно-обучающая система – URL: https://www.youtube.com/watch?v=3CDsMHzh-Co (дата обращения: 30.01.2019 г.)

98. Тренажерная система (стыковка) – URL: https://www.youtube.com/watch?v=0eVBXK0vivA (дата обращения: 30.01.2019 г.)

99. Kuravsky L.S., Popkov S.I., Artemenkov S.L. An applied multi-agent system within the framework of a player-centered probabilistic computer game. - International Journal of Modeling, Simulation, and Scientific Computing, Vol. 9, No. 1 (2018), 17 p, DOI: 10.1142/S1793962317500635 [SCOPUS].

100. Попков С.И. Применение и разработка тренажера для автоматизированных беспилотных летательных аппаратов и робототехнических комплексов на базе вероятностной модели поведения прикладной многоагентной системы. – Нейрокомпьютеры: разработка, применение, № 5, 2019, с. 5-17.

101. Kuravsky L.S., Popkov S.I. and Artemenkov S.L. Applied multi-agent system to study behavior of operators of complex technical systems. – In: Proc. First World Congress on Condition Monitoring 2017 (WCCM 2017) The International Society for Condition Monitoring (ISCM), British Institute of Non-Destructive Testing (BINDT). 2017 [SCOPUS].

102. Kuravsky L.S., Popkov S.I. Forecasting macro parameters representing the behavior of an applied multi-agent system. – International Journal of Modeling, Simulation, and Scientific Computing, 2018, Vol. 9, No. 6 (2018), 1850052, 15 pp, DOI: 10.1142/S1793962318500526 [SCOPUS].

103. Kuravsky L.S., Popkov S.I. Forecasting behavior of a stochastic multi-agent system. - In: Proc. 15th International Conference on Condition Monitoring & Machinery Failure Prevention Technologies, Nottingham, UK, September 2018. - 9 pp.

104. Куравский Л.С., Попков С.И. Представление общих закономерностей поведения многоагентной системы с помощью ее макропараметров (тезисы). XVI Всероссийская научная конференция «Нейрокомпьютеры и их применение». МГППУ, 2018. – 4 с.

105. Business Process Management – URL: https://www.bonitasoft.com/business-process-management-bpm (дата обращения: 30.01.2019 г.)