ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

На правах рукописи

Алимов Александр Александрович

Управление поведением многозадачных интеллектуальных агентов в системах реального времени

Специальность 05.13.01 —
«Системный анализ, управление и обработка информации (информационные технологии и промышленность)»

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
д.т.н., профессор кафедры САПРиПК
Кравец Алла Григорьевна

Волгоград — 2017

Оглавление

Введение

Мультиагентные системы реального времени являются эффективным инструментом для моделирования сложных процессов, в которых участвует большое количество активных автономных сущностей. К таким процессам относятся потоки городского движения, логистические системы, социальные явления, эпидемии. Методы мультиагентного моделирования используются также для поиска и обработки информации в информационных сетях, системах управления автономными роботами. Перспективным направлением развития мультиагентных систем является разработка беспилотных автомобилей и летательных аппаратов.

Одним из наиболее активно-развивающихся направлений является использование мультиагентного подхода в системах виртуальной реальности и видеоиграх. В этих системах агентное моделирование применяется для сбора статистики, расчета вычислительной нагрузки серверов и реализации игрового искусственного интеллекта.

В мультиагентных системах (МАС), которые работают в режиме реального времени, агент должен рациональным образом решить поставленные перед ним задачи с минимальными затратами ресурсов. В режиме реального времени использование высокоточных методов многокритериальной оптимизации затруднено, поэтому МАС обычно решают задачу приближенными методами с помощью проблемно-ориентированных эвристик. Однако, применение эвристических оптимизаций уменьшает общность модели МАС.

Проблемами разработки МАС являются ресурсоемкость используемых алгоритмов, высокая сложность и связанность моделей. Эти факторы негативно влияют на качество МАС, снижает их сопровождаемость и производительность, повышают стоимость разработки. Дополнительную сложность составляет динамический характер окружающей среды, в которой приходится функционировать агенту, так как состояние среды может существенно измениться за время принятия решения. Агент должен адаптироваться к таким изменениям за как можно меньшее время.

Таким образом задача разработки моделей и методов повышения качества МАС является актуальной.

Исследования и разработки в области мультиагентных систем начались в 70-80х годах XX века. Одна из самых ранних агентных моделей была предложена в работе Томаса Шеллинга для описания процессов сегрегации. Исследования МАС также изложены в монографиях Тарасова В.Б., Каляева И.А., Бугайченко Д.Ю. С развитием вычислительной техники появилось большое количество прикладных и исследовательских мультиагентных систем и средств агентного моделирования, таких как Swarm, NetLogo, RePpast и AnyLogic.

Для реализации МАС часто используют классические алгоритмы ИИ, предложенные Richard E.Fikes, Nils J. Nilsson и предназначенные для поиска в пространстве состояний решаемой задачи. Из-за ограничений вычислительных ресурсов такие системы часто прибегают к использованию квази-оптимальных методов, которые позволяют получить близкое к оптимальному решение за приемлемое время, Adi Botea, Martin Müller. Существенное влияние на развитие индустрии мультимедиа(видеоигр и графики) оказали работы C. Reynolds и предложенные им модели поведения стай(Flocking behavior, boids). Большое распространение получили вариации алгоритма A* разработанного R. Detcher и J. Pearl. Большое количество работ посвящено динамической корретировке перемещений агентов в пространстве: Javier Alonso-Mora, Jur van den Berg, Ming C. Lin, Dinesh Mnocha, Rahul Narin. В этой области большое значение имеет схожесть поведения ИИ с поведением человека, в связи с чем большое внимание уделяется генерации «человекоподобных» моделей поведения. МАС являются эффективным инструментом описания конкурентных процессов, таких как стратегические игры (Jeff Orkin) и моделирование военных действий (Новиков Д.А.).

Объектом исследования является мультиагентная система реального времени. **Предметом исследования** является управление поведением интеллектуального агента.

Целью данной работы является повышение качества мультиагентных систем, функционирующих в режиме реального времени в динамической недетерминированной среде.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. провести анализ проблем разработки мультиагентных систем реального времени;

2. исследовать существующие модели поведения агентов;
3. разработать модель поведения агента;
4. разработать программную архитектуру мультиагентной системы;
5. провести анализ критериев качества МАС, разработанной на основе предложенной модели.

Научная новизна: предложена формальная модель мультиагентной системы, в отличие от существующих, учитывающая реакцию окружающей среды на действие или бездействие агентов как интерактивных сущностей самой среды. Получены следующие новые научные результаты:

1. модель интеллектуального агента, процесс принятия решений которого, в отличие от известных, реализуется на основе расширенного дерева поведения.
2. метод динамического изменения структуры дерева поведения агента, отличающийся от известных реализацией продукционных правил для обработки событий, воспринимаемых агентом;
3. метод оценки приоритетов целей агента, отличающийся мультипликативной сверткой нормализованных частных эвристических критериев;
4. архитектура программной реализации МАС в виде фреймворка.

Практическая значимость заключается в предложенной модели поведения интеллектуального агента, разработанных алгоритмов и программном обеспечении. По сравнению со стандартным деревом поведения разработанная модель позволяет добиться более высокой адаптивности агентов при меньшей сложности дерева поведения. Предложенные модели реализованы в виде фреймворка – библиотеки классов, реализующих базовые компоненты агентов: интерфейсы сенсоров, приводов, память. Использование фреймворка позволяет повысить качество разрабатываемых МАС.

Методология и методы исследования. В диссертации использованы методы системного анализа, теории управления, математического моделирования, проектирования информационных систем, теория предельной полезности.

Основные положения, выносимые на защиту:

1. модель интеллектуального агента на основе расширенного дерева поведения, позволяющая адаптировать поведение агента к изменениям окружающей среды;

2. метод динамического изменения структуры дерева поведения агента, отличающийся от известных реализацией продукционных правил для обработки событий, воспринимаемых агентом;
3. метод оценки приоритетов целей агента, позволяющий агенту в режиме реального времени рационально выбирать наиболее важные цели;
4. архитектура программной реализации МАС в виде фреймворка, позволяющая повысить сопровождаемость МАС за счет уменьшения сложности и связности ее компонентов.

Достоверность полученных результатов подтверждается эксперементальными данными. Результаты находятся в соответствии с результатами, полученными другими авторами. Успешность практического применения результатов работы подтверждается актом внедрения.

Апробация работы. Основные результаты работы докладывались на: международной конференции по искусственному интеллекту (MIWAI-2015) и региональных конференциях. Реализация предложенных моделей и методов поддержана фондом содействия инноваций по программе «У.М.Н.И.К.»

Публикации. Основные результаты по теме диссертации изложены в 10 печатных изданиях, 5 из которых изданы в журналах, рекомендованных ВАК, 1 работа в зарубежном издании, индексируемом в базе научного цитирования Scopus, 3 — в тезисах докладов, 1 свидетельство о регистрации программ для ЭВМ.

Глава 1. Анализ проблем разработки мультиагентных систем

1.1 Определение понятия «агент» и область применения

Парадигма агентно-ориентированного моделирования является развитием объектно-ориентированной парадигмы, в которой агенты являются моделью некоторой активной сущности в моделируемом процессе. Ключевое отличие агентно-ориентированного подхода заключается в том, что агенты являются *субъектами* в процессе моделирования и обладают желаниями, намерениями и способностью выполнять действия, в то время, как в объектно-ориентированный подход предполагает, что программа осуществляет операции над *объектами* [1]. Из-за сходства подходов значительная часть существующих средств агентного моделирования использует лингвистическое обеспечение, основанное на объектно-ориентированной парадигме.

На сегодняшний день не существует однозначного и точного определения термина «агент». В этой работе будет использовано определение, данное Расселом и Норвигом: «агент – это активная сущность, которая воспринимает окружающую среду с помощью сенсоров и модифицирует ее, выполняя действия с помощью эффекторов (приводов). Функцией агента называется функция отображающая последовательность восприятий в последовательность выполняемых действий»[2]. Реализация функции агента называется программой агента.

Исследования и разработки в области мультиагентных систем начались в 70-80х годах XX века. Одна из самых ранних агентных моделей была предложена в работе Томаса Шеллинга [3]. Обширное исследование мультиагентных систем представлено в монографии Тарасова [4]. С развитием вычислительной техники появилось большое количество прикладных и исследовательских мультиагентных систем и средств агентного моделирования, таких как Swarm [5], NetLogo[6], RePpast и AnyLogic. Эти инструменты предназначены для моделирования бизнес-процессов с целью их оптимизации.

Благодаря присущему агентам свойству автономности, агентное моделирование нашло применение в робототехнике [7], позволяя децентрализовать систему управления группами роботов. В отличии от прочих методов имитационного моделирования, агентное моделирование фокусирует внимание на индивидуальных свойствах каждого агента, благодаря чему упрощается модели-

рование человеческого поведения[8; 9]. С использованием МАС может также осуществляться моделирование военных действий [10].

Мультиагентные системы нашли свое применение в области разработки видеоигр[11—13], для моделирования поведения игроков[14]. В игровой индустрии МАС используется для организации нагрузочного тестирования, реализации игрового ИИ и моделирования природных эффектов. Цели и задачи игрового, прикладного и академического искусственного интеллекта а значительно отличаются друг от друга. Очень часто в игровом ИИ критерий оптимальности отодвигается на второй план. Вместо этого исследователи и разработчики игрового ИИ концентрируются на создании иллюзии достоверности поведения персонажей в условиях ограниченных вычислительных ресурсов[15; 16].

Научно-исследовательские и коммерческие системы разработки МАС представляют собой конструктор, позволяющий пользователю собрать МАС из заранее заготовленных компонентов, определяющих роли агентов в системе. Многие инструменты общего назначения, такие как Anylogic, предлагают пользователю реализовать программу на языке Java с использованием элементов визуального программирования.

В соответствии с определением агента, его поведение может быть представлено как функция отображающее воспринятую агентом информацию в изменение состояния агента. В общем виде структура агента представлена на рисунке 1.1.

Как правило, интеллектуальный агент состоит из следующих компонентов: сенсоров, приводов, памяти и модели поведения(программы агента). Сенсоры и приводы являются интерфейсом для взаимодействия между агентом и окружающей средой. В памяти сохраняются результаты восприятия окружающей среды сенсорами. В некоторых реализациях МАС, например на основе марковских цепей, память агента может отсутствовать. Программа агента реализует его модель поведения и является наиболее сложным компонентом. Внутренне представление программы агента в первую очередь зависит от задачи, решаемой МАС.

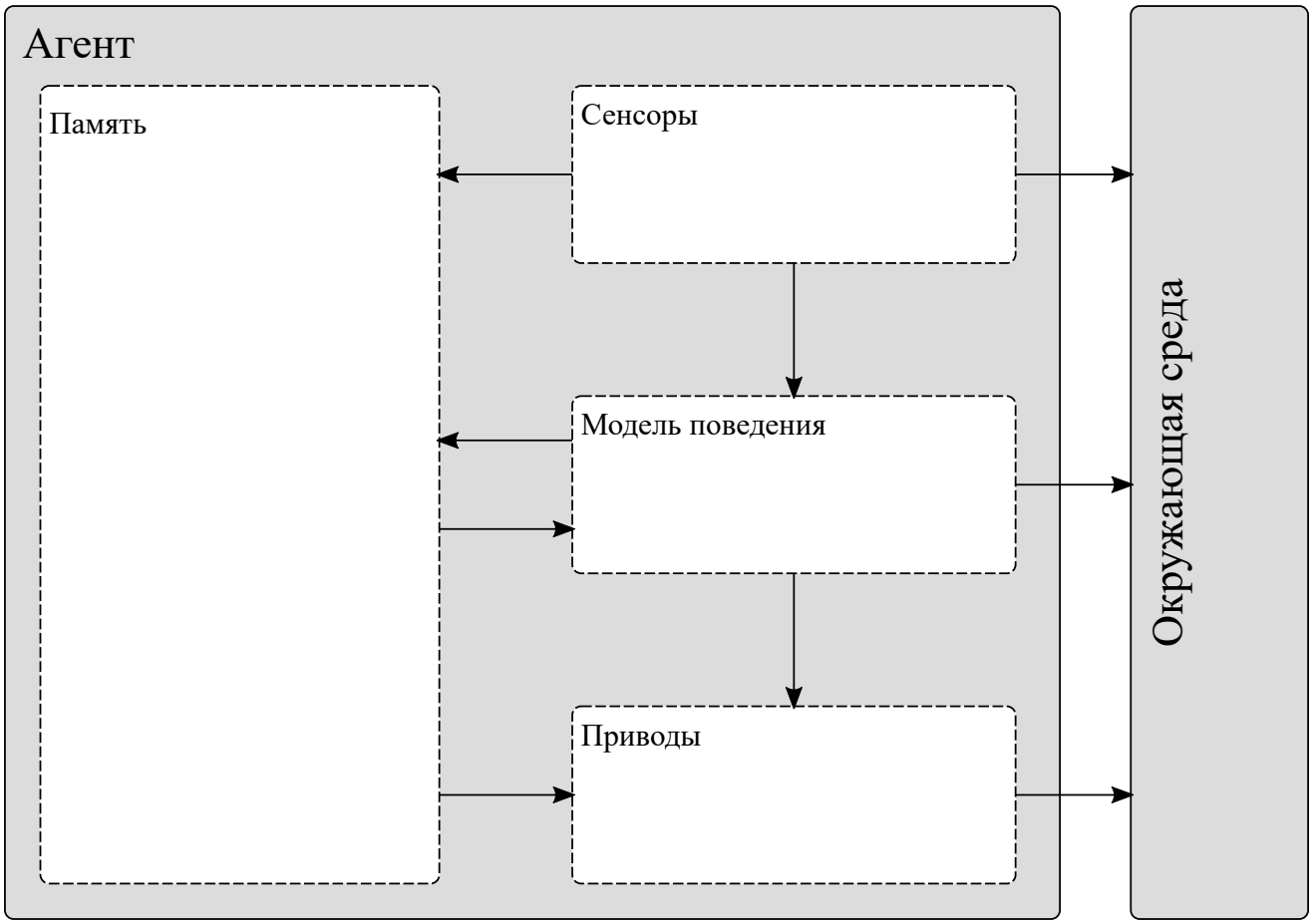

Рисунок 1.1 — Обобщенная структура агента

1.2 Классификация мультиагентных систем

Свойства модели агента определяются решаемой задачей, формирующей окружающую среду в которой функционирует агент. Можно утверждать, что в первую очередь они зависят от характера окружающей среды, в которой он функционирует. Выделяют следующие классификационные признаки МАС [2]:

- количество агентов в системе;
- обозреваемость окружающей среды;
- характер отношений между агентами;
- модель представления времени;
- дискретность или непрерывность пространства состояний;
- наличие у агента знаний о законах функционирования окружающей среды;
- детерминированность переходов между состояниями объектов МАС;
- динамичность;

В зависимости от *количества агентов* различают системы с единственным агентом и с множеством агентов. Системы с единственным агентом являются вырожденным случаем и, как правило, на представляют интереса, так как могут быть заменены объектно-ориентированной программой.

По признаку обозреваемости среды выделяют системы с *полностью или частично-обозреваемой средой*. В полностью обозреваемой среде агенту в каждый момент времени точно известно состояние всей системы. Большинство прикладных задач представляют собой частично-обозреваемую среду. Применительно к игровому ИИ обозримость среды очень высока, так как значительная часть информации, необходимой для принятия решений может быть извлечена непосредственно из среды моделирования. В то же время, ограничения вычислительных ресурсов снижают эту определенность

Отношения между агентами могут быть *кооперативными или конкурентными*. Существуют также системы, в которых отношения между агентами могут динамически изменяться в течении времени либо полностью отсутствовать.

Модель представления времени может быть *эпизодической или последовательной*. В эпизодической среде действие агента зависит только от текущего состояния входов, в последовательной – от результатов предыдущих действий..

Пространство состояний является множеством всех допустимых состояний системы и может быть *непрерывным или дискретным*. В рассматриваемом классе задач пространство состояний формируется непрерывными (координаты, вращение) и дискретными величинами, в следствии чего является кусочным. Обычно для решения задач в непрерывном пространстве состояний, как и в случае с моделью времени, применяют дискретизацию и используют соответствующие алгоритмы для его обработки.

Знания о законах функционирования системы позволяют агенту предсказывать изменения состояния системы и более эффективно принимать решения. Как правило, в любой модели агент изначально обладает частичными знаниями об окружающей среде. В противном случае для возможности действовать рационально агенту необходимо эти знания получить, например, в результате процедуры обучения. В рассматриваемом классе задач агенту достаточно полно осведомлен о законах функционирования окружающей среды.

В зависимости от предсказуемости результатов действий агента система может быть *детерминированной* или *стохастической*. Как правило игровой

ИИ представляет собой стохастическую систему, в которой на результат действия серьезное воздействие оказывают случайные факторы. В целях упрощения можно считать, что из всех возможных исходов выполнения действия один является успешным, именно к такому результату агент стремится. Остальные исходы действий агента будет считать неудачными.

Различают системы со *статической и динамической* окружающей средой. В статической среде агенты принимают решение до того, как состояние среды изменится. Неизменность состояния является гарантированной. В динамической окружающей среде за время принятия решения агентом состояние среды может существенно измениться. Системы реального времени как правило являются динамическими, так как моделирование процесса принятия решений может осуществляться значительное время параллельно с моделированием физических процессов.

При решении задач игрового ИИ, управления автономными роботами и беспилотными аппаратами окружающая является динамической, известной, частично обозреваемой, стохастической, непрерывной и последовательной. В процессе упрощения модели окружающая среда подвергается дискретизации с несколькими уровнями детализации. Алгоритмы планирования и принятия решения работают довольно продолжительное время, за которое состояние окружающей среды может измениться, например, объекты могут переместиться в пространстве. Для данного класса задач актуально решение проблем, связанных с высоким временем отклика при принятии решений.

1.3 Классификация задач агента

Современные мультиагентные системы как правило являются проблемно-ориентированными. Отказ от обобщенных вычислений позволяет применять различные эвристики, основанные на особенностях решаемой задачи, в целях повышения производительности и повышения качества принимаемых решений.

В рассматриваемом классе МАС агенты обладают способностью перемещаться в пространстве и взаимодействовать с различными объектами окружающей среды, схема которой представлена на рисунке 1.2.

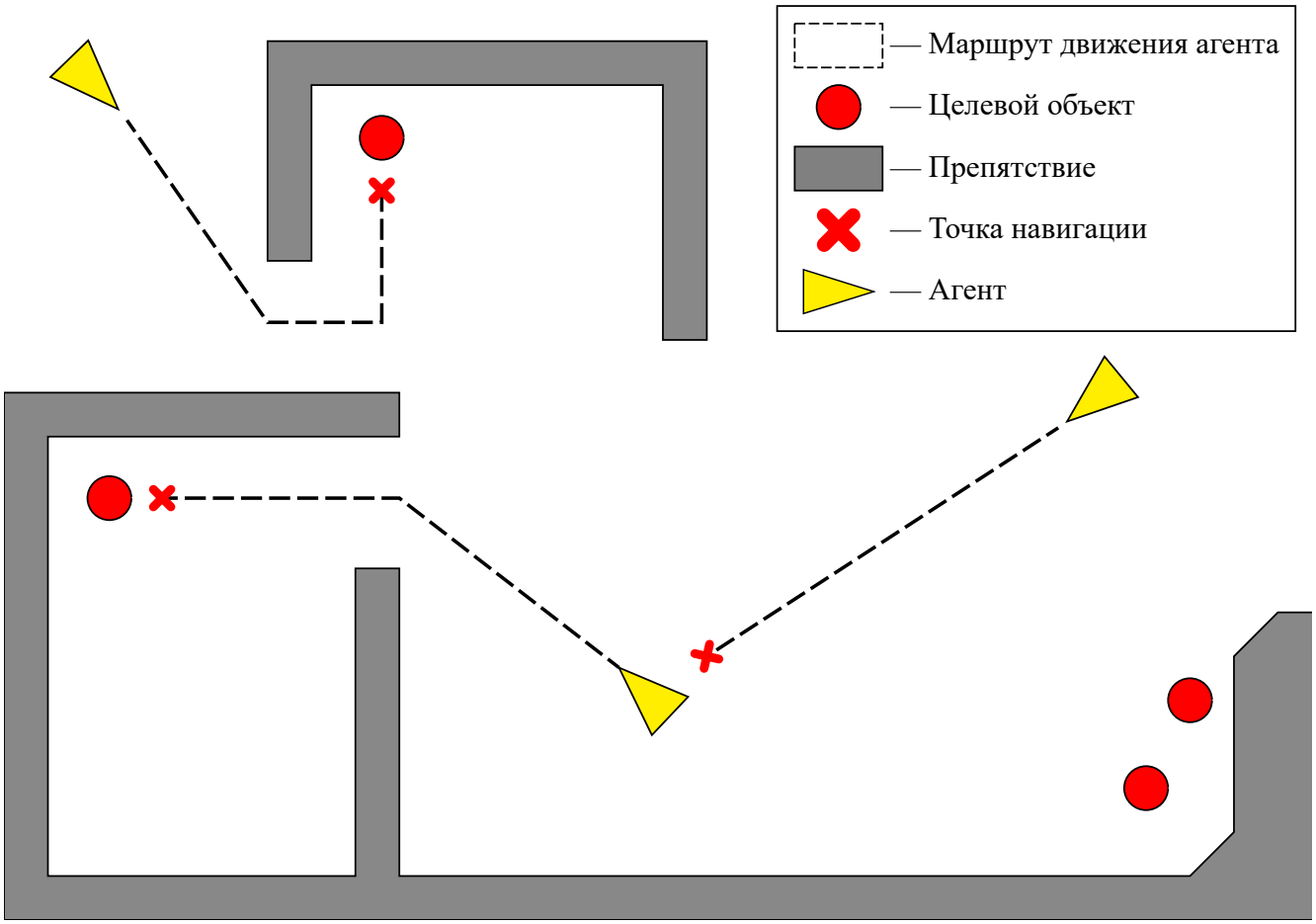

Рисунок 1.2 — Окружающая среда и объекты в мультиагентной системе

Для успешного функционирования агент должен решить ряд задач, направленных на максимизацию получаемой полезности. Такими задачами являются:

- навигация и перемещение в пространстве окружающей среды, включая планирование маршрута и избегание препятствий;
- целенаправленное воздействие на объекты окружающей среды;
- принятие решений и планирование действий;
- адаптация к изменениям окружающей среды;

При моделировании поведения движимых объектов реального мира, таких как пешеходы, транспорт или беспилотные аппараты, необходимо «научить» агента ориентироваться в трехмерном пространстве и прокладывать маршруты. Выбор маршрута должен осуществляться таким образом, чтобы рационально использовать ресурсы агента и безопасно достигать желаемой области в пространстве. При движении по маршруту агент должен уметь избегать столкновений с динамическими препятствиями – другими объектами окружающей среды.

Задача планирования действий схожа с задачей планирования перемещений. Агенту необходимо построить последовательность действий, которые необходимо выполнить для того, чтобы оказаться в желаемом состоянии. Перемещения в 3-х мерном пространстве обычно отделены от прочих действий с целью обеспечения возможности использования оптимизаций, применимых к 3-х мерному пространству.

Планирование действий может осуществляться на нескольких уровнях. Обычно выделяют стратегический и тактический уровни планирования. На стратегическом уровне осуществляется управление группами агентов.

Состояние динамической окружающей среды может измениться в процессе построения и выполнения агентом плана действий. Для того, чтобы план действий агента не терял актуальность агенту необходимо немедленно реагировать на воспринимаемые события. Мгновенные реакции агента должны быть направлены на адаптацию модели поведения к изменениям. На сегодняшний день существуют различные методы адаптации. Некоторые из них предполагают изменение состояния памяти агента [17]. Новые факты и знания, полученные в ходе моделирования, позволяют более точно решать поставленные задачи. В [18] предлагается метод адаптации структуры агента на основе правил. Структурные изменения отражаются на изменении модели поведения и позволяют агенту более эффективно решать поставленные задачи. Другие способы адаптации реализуют обучение с подкреплением и эволюционные алгоритмы [19] непосредственно на этапе моделирования. Использование данного метода оправдано в длительных процессах моделирования, в которых агент успевает на практике оценить результаты своих действий.

Адаптация модели поведения агентов может осуществляться на стратегическом уровне, в этом случае процесс адаптации затрагивает структуру связей между агентами. В [20] преложен метод генерации модели поведения игрока в стратегиях реального времени в соответствии с навыками оппонента-человека на основе метода эволюционной оптимизации.

В мультиагентной системе агенты являются автономными сущностями, но при этом их деятельность во многом определяется характером взаимодействия агентов между собой. Коммуникация между агентами обычно осуществляется посредством специализированного протокола сообщений[21; 22]. Сообщения в агентно-ориентированной системе схожи с сообщениями, которые ис-

пользуются при построении объектно-ориентированных систем. Отличие заключается в меньшей строгости правил обработки сообщений. В объектно-ориентированной системе отправка сообщения обычно реализуется как вызов императивного метода с определенным побочным эффектом, в то время как в агентно-ориентированной системе агент реагирует на сообщение в соответствии с собственными намерениями. Посредством сообщений агенты могут делегировать друг-другу задачи и передавать информацию.

Одной из ключевых проблемам коммуникации между агентами являются формат представления сообщений и механизм их рассылки[23]. От выбранной модели коммуникации зависит общая эффективность мультиагентной системы.

Использование модульной архитектуры позволяет инкапсулировать решение рассмотренных задач агента в отдельные слабо-связанные модули. Входные и выходные данные каждой задачи определяют интерфейсы взаимодействия модулей между собой. Как правило, одна из рассмотренных задач является определяющий в формировании модели поведения агента, например управление беспилотным аппаратом базируется на решении задачи навигации и избегания препятствий, моделирование распространения вирусных заболеваний – на задаче коммуникаций. В рассматриваемом классе МАС ключевыми задачами является планирование действий и принятие решений.

1.4 Критерии качества мультиагентной системы

Для анализа существующих методов решения и проблем в области разработки мультиагентных систем следует выделить критерии качества мультиагентной системы. Существующие МАС можно классифицировать на программные и программно-аппаратные системы. В программно-аппаратных МАС ключевым компонентом обычно является программное обеспечение, реализующее логику поведения МАС.

Рассмотрим жизненный цикл программной системы. Для большинства программ характерно наличие в жизненном цикле следующих этапов:

1. формирование и анализ требований;
2. проектирование;

3. реализация;
4. тестирование;
5. эксплуатация.

Перечисленные этапы могут быть скомпонованы в водопадную модель или многократно повторяться в спиральной модели. Все этапы можно условно разделить на два периода: разработку и выполнение. На каждом этапе жизненного цикла являются важными определенные характеристики системы.

На ранних итерациях разработки системы происходит частое изменение требований к системе, что приводит к необходимости пересмотра реализованных моделей агентов. Чем более фундаментальными являются изменения модели, тем больше они влияют на изменение стоимости всего проекта. Исходя из этого, важным качественным показателем модели агента является сложность адаптации модели к изменению функциональных требований.

Очень часто в рамках одной системы необходимо разработать несколько вариаций поведения агентов для выполнения различных между собой функций. В этом случае выгодно иметь модульную систему, в которой функциональность агента может быть составлена из слабо-связанных модулей. Идеальная модель агента должна оперировать объектами любой структуры с произвольными зависимостями, тогда изменение функциональных требований отразится лишь на содержимом базы знаний агента. Первые интеллектуальные программы, такие как «универсальный решатель теорем», работали именно таким образом, они обладали высокой степенью обобщенности и оперировали абстрактными фактами, однако алгоритмы, используемые в этих программах, обладали чрезвычайно высокой вычислительной сложностью, обусловленной своей комбинаторной природой. Для того, чтобы интеллектуальная система могла выполнить свою работу за приемлемое время, необходимо максимально сузить рабочее пространство системы.

Таким образом, в мультиагентной системе реального времени с множеством функциональных ролей необходимо добиться компромисса между производительностью, точностью, модульностью и сопровождаемостью системы.

Критерии оценки качества программных систем регламентированы международными стандартами ISO [24] и национальными стандартом ГОСТ[25]. Мультиагентные системы являются программными или программно-аппаратными системам, поэтому критерии качества программных систем могут

быть успешно адаптированы для оценки МАС [26]. Стандарты декларируют следующие критерии качества програмного обеспечения:

1. функциональные возможности;
2. надежность;
3. практичность;
4. эффективность;
5. сопровождаемость;
6. мобильность.

Функциональные возможности – это набор атрибутов, определяющих степень удовлетворения программным продуктом потребностей пользователя. Применительно к МАС критерий оценивает способность агента к решению поставленной задачи, выполнению рациональных действий. Функциональные возможности МАС определяется адекватностью модели поведения агента. Чем выше показатели результативности агентов, тем выше качество МАС.

Надежность определяет способность программы сохранять свой уровень качества во время функционирования. Надежность МАС определяется способностью агентов сохранять рациональное поведение в случае изменений окружающей среды. Можно говорить о том, что надежность МАС определяется адаптивностью и устойчивостью модели поведения агента.

Практичность определяет объем работ, требуемых для использования ПО. Соответственно практичность МАС характеризуется объемом работ, необходимым для реализации требуемого функционала в рамках выбранной модели поведения агента. Чем меньше настраиваемых параметров имеет модель поведения агента, тем выше практичность МАС.

Эффективность ПО определяется соотношением между уровнем качества функционирования ПО и объемом используемых ресурсов. В контексте МАС эффективность определяется количеством вычислительных ресурсов, потребляемых агентом в процессе принятия решений, в единицу времени. Эффективность МАС будет тем выше, чем меньше данных требуется хранить в оперативной памяти и чем меньше операций будет над этими данными проводиться. Количество ресурсов, потребляемых одним агентом определяется вычислительной сложностью используемых алгоритмов.

Сопровождаемость является комплексным критерием, оценивающим объем работ, необходимых для внесения конкретных модификаций в ПО.

Аналогично практичности, сопровождаемость модели агента определяется объемом работ необходимых для внесения изменений в реализацию агента: изменение знаний об окружающей среде, обучение выполнению новых действий;

Мобильность – набор атрибутов, определяющих возможность переноса программного обеспечения из одного окружения в другое. Мобильностью модели агента будем считать возможность переноса и повторного использования компонентов поведения агента.

Наиболее формализуемый критерий из перечисленных – эффективность, может быть замерена непосредственно на физическом уровне. Замеры учитывают время выполнения методов программы агента и объем оперативной памяти, используемый соответствующим процессом.

На основе критериев качества МАС сформируем критерии оценки моделей поведения агента. В [27] в качестве критериев оценки качества МАС предлагаются адекватность модели времени, причинность (causality) и воспроизводимость (reproducibility). В [28] основной характеристикой качества МАС считается способность агентов к социальному взаимодействию: коммуникации (Communication), кооперации (Cooperation), соглашению (Negotiation).

По результатам анализа был сделан вывод о том, что качество мультиагентной системы в первую очередь определяется качеством модели управления поведением агента. Для оценки качества выбраны следующие критерии:

– адекватность модели;
– сопровождаемость;
– производительность;
– адаптивность.

Адекватность модели поведения агента определяется тем, насколько эффективно агент может выполнять свои функции. Чем выше адекватность модели, тем меньше отклонение результатов, полученных агентом от оптимальных. С точки зрения утилитарного подхода адекватность модели можно количественно оценить суммарной полезностью, которую агент получает в процессе моделирования. Измерение результативности может измеряться напрямую, если известна целевая функция агента, или косвенно, в этом случае функция полезности определяется экспертом.

Сопровождаемость модели поведения агента – это комплексная характеристика, оценивающая возможность внесения изменений. Модель поведения

агента является программой, поэтому методы оценки сложности программ [29] применимы и для моделей поведения агентов. На сопровождаемость негативное влияние оказывает объем модели – количество элементов, классов, состояний, строк исходного кода (SLOC). Значительное влияние на сопровождаемость модели оказывает цикломатическая сложность используемых алгоритмов и связность модели[30; 31].

Производительность является характеристикой реализации модели и определяется временем, необходимым для принятия решения агентом. На производительность оказывают влияние особенности реализации модели, алгоритмическая сложность используемых алгоритмов и инфраструктурные расходы на организацию функционирования окружающей среды.

Адаптивность модели поведения – это способность агента сохранять желаемый уровень производительности при значительном изменении параметров окружающей среды.

1.5 Задача перемещения агента в окружающем пространстве

Наиболее часто рассматриваемой задачей в агентно-ориентированном моделировании является задача планирования маршрута. Ее следует отделить от остальных задач, как наиболее формализованную и проработанную. Большинство работ в области игрового ИИ посвящено исследованию методов поиска маршрута перемещения.

Знание особенностей алгоритмов планирования и осуществления перемещений агента позволяет разработать эффективный в плане производительности и поддерживаемый интерфейс для взаимодействия компонента навигации с другими модулями агента. Для решения задачи планирования перемещений используются алгоритмы работающие с навигационными графами. Пространство окружающей среды разбивается на области, так что внутри каждой из них для любой пары точек внутри области возможно перемещение по прямой. Соседние области связываются переходами – ребрами графа.

Существуют различные вариации навигационных графов, наиболее часто используются:

– граф, состоящий из маршрутных точек (Way-point graph);
– регулярная навигационная сетка;
– навигационная сетка, повторяющая геометрию окружающей среды;

Для разбиения пространства окружающей среды может быть использована регулярная или иррегулярная сетка. При построении регулярной сетки наиболее часто используется квадратное или гексагональное разбиение. Недостатком такого метода является избыточное количество вершин графа и, как следствие, увеличенное время поиска. В целях оптимизации ячейки регулярной сетки, достижимые по прямой, могут быть объединены

Построение маршрута может может осуществляться при помощи классических алгоритмов поиска в графе (поиск в глубину и поиск в ширину, алгоритм Дейкстры), однако знание свойств окружающей среды позволяет использовать более эффективные специализированные алгоритмы и эвристики. Выбор алгоритма зависит от свойств навигационной сетки и полноты информации.

В случаях, когда структура навигационной сетки известна наиболее эффективными являются алгоритм A* [32] и его вариации. Этот алгоритм получил наибольшее распространение в индустрии разработки видео-игр.

Для больших пространств применяют иерархические алгоритмы, в которых поиск осуществляется последовательно в графах с различным уровнем детализации [33; 34]. Граф верхнего уровня покрывает все пространство и является наименее детализированным. Вложенные графы покрывают пространство соответствующее одному или нескольким узлам родительского графа и имеют более высокую детализацию. При определенных условиях могут быть применены эвристики, уменьшающие количество итераций поиска, как это сделано в алгоритмах Jump Point Search [35; 36].

Если окружающая среда не меняется, то план может выполнятся в виде последовательности элементарных действий, в противном случае план необходимо корректировать в соответствии с изменением окружающей среды. Для прикладных задач как и для видео-игр характерно наличие постоянно изменяющейся динамической окружающей среды – объекты могут быть перемещены или уничтожены. Наличие другого агента в определенной точке пространство так же может влиять на возможность перемещения. Однократное планирование маршрута в такой системе не является достаточным, поэтому агент вынужден

постоянно корректировать свой маршрут, при этом план может быть перестроен полностью или частично.

Существенное влияние на развитие агентного моделирования в области индустрии мультимедиа(видеоигр и графики) оказали работы Крейга Рейнольдса и предложенные им модели поведения стай(Flocking behavior, boids) [37; 38]. Управление агентом осуществляется за счет изменения направления и скорости при приближении агента к ключевым точкам или препятствиям. Модель Steering Behaviour позволяет комбинировать элементарные шаблоны поведения, такие как прибытие, следование, удаление в более сложные поведенческие сценарии, включающие в себя перемещение в группе, избегание препятствий. Композиция элементов поведения осуществляется на основе взвешенной суммы управляющих воздействий. Существующие реализация алгоритмов, такие как OpenSteer [39], имеют ряд проблем мобильности, заключающихся в высокой связанности алгоритмов и внешних систем: рендеринга, физики и других. Связанность возникает из-за необходимости получения агентом данных, в том числе и о своих координатах, от внешней среды.

Алгоритмы избегания столкновений, основанные на скорости сближения[40; 41] расширяют модель управления Steering Behavior. Агент ограничивает свою скорость таким образом, чтобы при перемещении не столкнуться с другим агентом. Эта группа алгоритмов требует определять возможность столкновения для любой пары объектов каждую итерацию моделирования.

Принципиально иной подход – это представление агентов МАС в виде непрерывного потока (Continuous Crowds)[42]. Для каждого агента вектор желаемой скорости движения корректируется в соответствии с уравнением описывающим несжимаемую жидкость.

Для сглаживания маршрута передвижения компания Valve в серии игр Left4Dead применяла алгоритм реактивного следования по маршруту[43], сглаживающий перемещения по навигационному графу и позволяющему избегать столкновений с небольшими препятствиями, не представленным в графе. Алгоритм поиска маршрута возвращает массив точек, представляющий собой ломанную линию. Каждую итерацию агент выбирает целевую точку из массива, в которую он будет двигаться. Выбирается точка с максимальным индексом в массиве такая, что отрезок, соединяющий текущую точку с координатами агента и выбранную точку не пересекает ни одного препятствия.

Рассмотренные методы решения задачи навигации позволяют определить интерфейс для подсистем управления перемещениями агента. Входными данными подсистемы является область пространства окружающей среды, в которую необходимо попасть агенту. Выходными данными – маршрут движения агента и текущий вектор перемещения. Также подсистема управления перемещениями должна иметь доступ к сведениям о взаимном расположении объектов среды.

1.6 Конечно-автоматные модели поведения агентов

Поведение агента может быть описано конечным автоматом, в котором состояния автомата описывают действия, выполняемые агентом. Конечно-автоматные модели получили широкое распространение в силу простоты реализации и наглядности представления и удобства описания сценариев поведения. Модель агента на основе конечного автомата может быть представлена кортежем:

$$a_{fsm} = \langle E_v, S_a, q_0, S_F, \delta_s \rangle \tag{1.1}$$

где:

E_v – множество входных символов;
S_a – множество состояний автомата;
$S_F \subset S_a$ – множество конечных состояний;
$s_0 \in S_a$ – начальное состояние;
$\delta_s : E_v \times S_a \to S_a$ – функция перехода между состояниями.

Множество E_v входных символов конечного автомата, соответствует событиям, которые агент воспринимает из окружающей среды. События составляют основу цикла моделирования. Простейшие модели поведения могут напрямую отражать воспринимаемое событие в следующее действие агента. Агенты, построенные по этому принципу называются «реактивными». Более сложные агенты используют для выбора действий дополнительные знания об окружающей среде, хранящиеся в памяти агента.

Множество состояний конечного автомата S_a соответствует действиям, которые может выполнять агент. Функционирование агента начинается из начального состояния s_0.

Выбор агентом следующего действия описывается функцией перехода δ_s, которая соответствует программе агента. Функционирование агента продолжается до тех пора, пока он не окажется в конечном состоянии $s_f \in S_F$.

Конечные автоматы были доминирующей техникой в разработке игрового ИИ до конца 2000-х годов. Различные вариации автоматов использовались для моделирования действий персонажей игр и контроля анимации [44—46]. Причиной популярности конечно-автоматной модели стала удобная форма визуального представления в виде графа состояний, представленного на рисунке 1.3. Наглядность и структурированность модели обеспечивала хорошую *сопровождаемость* программной реализации по сравнению с классическими алгоритмами.

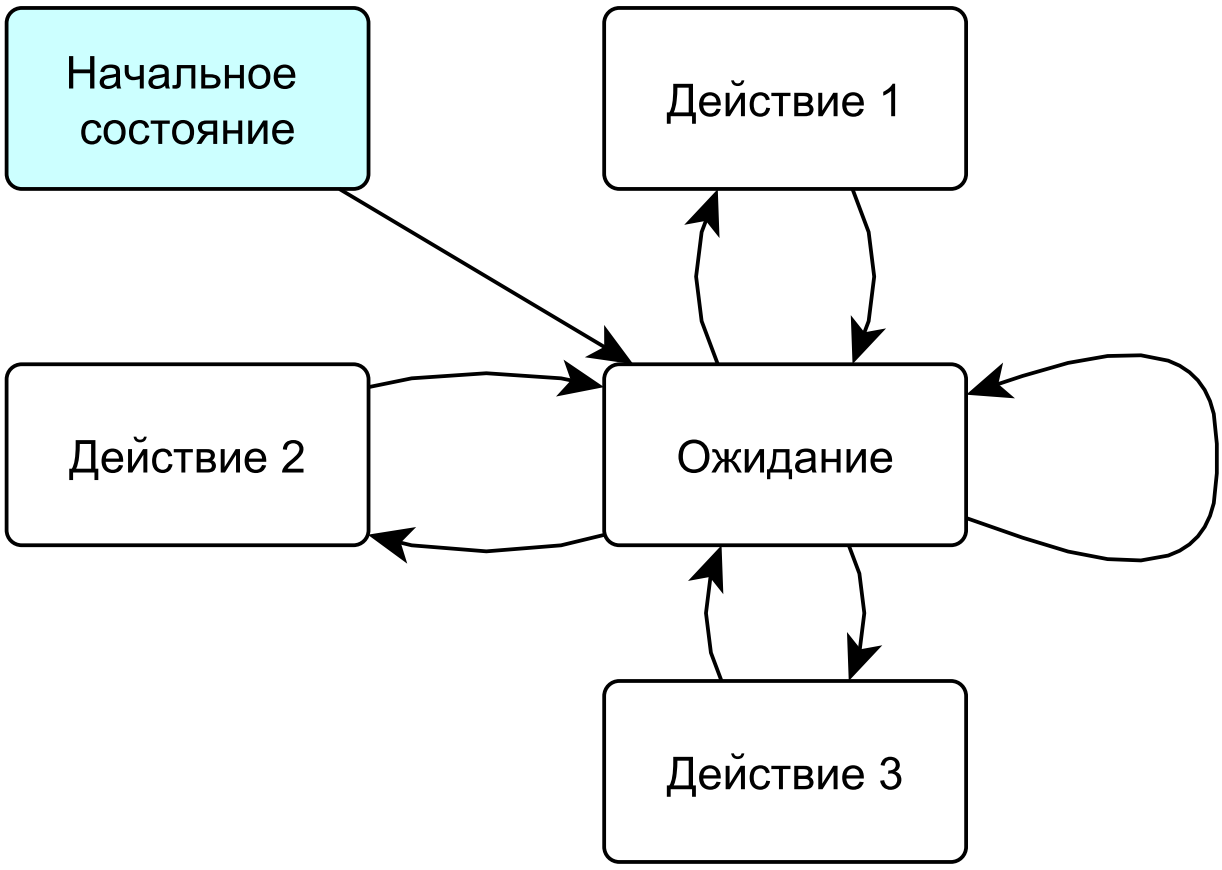

Рисунок 1.3 — Конечно-автоматная модель поведения

С необходимостью реализовывать все более сложное поведение росла сложность конечных автоматов. Это привело к тому, что разрабатываемые модели уже не были такими простыми и наглядным, как раньше [47]. Чем больше действий агент может выполнить, тем более сложным становится автомат. Добавление нового действия требует от разработчика модификации функции переходов таким образом, что необходимо пересмотреть все существующие связи между элементами автомата. Сложность добавления одного действия составит таким образом $N \cdot M$, где N – количество состояний в автомате, а M – количество переходов.

Таким образом с ростом вариативности поведения агента существенно повышается связанность конечно-автоматной модели и снижается ее сопровождаемость.

Для того, чтобы справится с ростом сложности конечного автомата были разработаны расширения модели – иерархические и конкурентные (параллельные) конечные автоматы.

Параллельные автоматы разделяют поведение агента на независимые слои. Каждый слой параллельного автомата может исполняться независимо либо в полной или частичной синхронизации с другими слоями. Данная модель может быть использована в том случае, если агент обладает способность. одновременного выполнения нескольких действия с помощью независимых приводов. Примером параллельного автомата является граф анимации персонажа, в котором один слой отвечает за анимацию перемещений, а другие за жестикуляцию и мимику.

Состояния иерархического конечного автомата (HFSM) могут содержать вложенные автоматы. Как правило функция перехода иерархического автомата может допускать переходы исключительно между состояниями, имеющими общего предка. Исполнение программы осуществляется от верхнего уровня к нижнему. При достижении конечного состояния осуществляется переход на уровень родительского состояния. Однако, существуют реализации допускающие произвольные переходы.

Деревья поведения являются развитием иерархической автоматной-модели. Концепция деревьев поведения была предложена и развита R.G. Dromey в серии работ [48—51] посвященных анализу требований и проектированию систем. Деревья поведения могут применяться для широкого ряда

задач: описания пользовательских сценариев, управления автономными роботами [52]. В процессе разработки моделей поведения интеллектуальных агентов обычно используется несколько модифицированная форма деревьев поведения [53; 54].

Дерево поведения является ориентированным деревом(графом) с тремя типами узлов: корневым, контрольным, исполняющим. Исполняющий узел не имеет дочерних узлов, корневой узел не имеет родительского узла.

Процесс выполнения дерева заключается в передаче сигнала от корневого узла дочерним. Каждый узел самостоятельно регулирует дальнейшее распространение сигнала в зависимости от типа узла. Узел может находиться в одном из следующих состояний: исполнение(Running), успех(Success), неудача(Failure), не инициализирован(None).

В зависимости от реализации контролирующие и исполняющие узлы могут различаться по типам. Наиболее типичные реализации включают в себя следующие контролирующие узлы:

– селекторы;
– последовательности;
– параллельные узлы;
– декораторы;

Селекторы передают управление одному из дочерних узлов в зависимости от состояния переменной в памяти агента. С помощью селекторов реализуются ветвления и альтернативные модели поведения. Последовательности осуществляют передачу управления дочерним узлам друг за другом до тех пор, пока все они не закончат работу; Параллельные узлы передают управление каждому дочернему узлу, имитируя параллельное выполнение. Декораторы меняют поведение единственного дочернего узла. С помощью декораторов могут быть организованы циклы или инверсия селектора.

Исполняющие узлы контролируют выполнение агентом действий, иногда до уровня локомоций включительно. Дерево поведения может одновременно контролировать не только поведение агента, но и процесс визуализации, синхронизируя выполняемые действия с анимацией. Подобный подход реализован в Unreal Engine 4[55], фреймворке RAIN AI [56].

Деревья поведения обладают значительно более высокой сопровождаемостью, чем классические конечно-автоматные модели за счет большей структу-

рированности и большего количества слоев абстракций. По сравнению с обычным автоматом граф модели не имеет циклов, узлы дерева строго типизированы в соответствии с используемой реализацией. Дерево поведение допускает повторное использование отдельных поддеревьев, таким образом обеспечивается модульность системы[57; 58].

Расширения модели увеличивают сопровождаемость за счет снижения количества информации, используемой для принятия решений. В случае дерева поведения селектор должен «знать», какому узлу отдать предпочтение, имея минимум информации о дочерних узлах. В некоторых реализациях селектор выбирает первый узел, выполнение которого завершается успешно.

Конечно-автоматные модели обладают очень низкой *адаптивностью*, так как фактически являются статическим алгоритмом обработки входных сигналов. Все возможные сценарии поведения должны быть изначально заложены в модель. Для частичного решения проблемы адаптивности разработаны динамические деревья поведения [59], которые используют приоритеты узлов, вычисляемые в процессе моделирования. Интересным подходом к построению моделей поведения являются эволюционные деревья поведения[60]. Структура дерева генерируется в процессе выполнения генетического алгоритма. Данный подход позволяет улучшать структуру дерева на протяжении всего процесса моделирования, однако требует значительных вычислительных затрат.

Приоритет узла может расчитываться исходя из полезности выполнения желаемого действия. Расчет приоритета инкапсулируется в узле дерева, что не приводит к снижению сопровождаемости.

В целом конечно-автоматные модели обладают высокой *производительностью*, так как не требуют больших объемов памяти и процессорного времени. В большинстве случаев время принятие решения слабо зависит от количества возможных состояний автомата N. В худшем случае, если возможны переходы в любое состояние, зависимость принимает вид $O(N)$.

Узким местом конечно-автоматной модели является процесс обработки входных сигналов. Для принятия решения агенту необходимо определять взаимное расположение объектов окружающей среды, классифицировать ситуации. Эта проблема является общей и распространяется не только на конечно-автоматные модели. В качестве решения обычно применяют кэширование результатов и проблемно-ориентированные оптимизации.

1.7 Целе-ориентированные модели поведения агентов

Одним из ключевых свойств интеллектуального агента является активность. Агент должен принимать решения которые способствуют достижению его целей. Под *целе-ориентированными* моделями поведения агентов понимают модели, которые в явной форме описывают целевое состояние и знания о законах функционирования окружающей среды. Целе-ориентированные модели позволяют построить цепочку рассуждений в пространстве знаний об окружающей среде.

Поведение целе-ориентированного агента может быть представлено последовательностью переходов в графе состояний окружающей среды. Каждый переход соответствует одному действию агента. Пример такого графа представлен на рисунке 1.4.

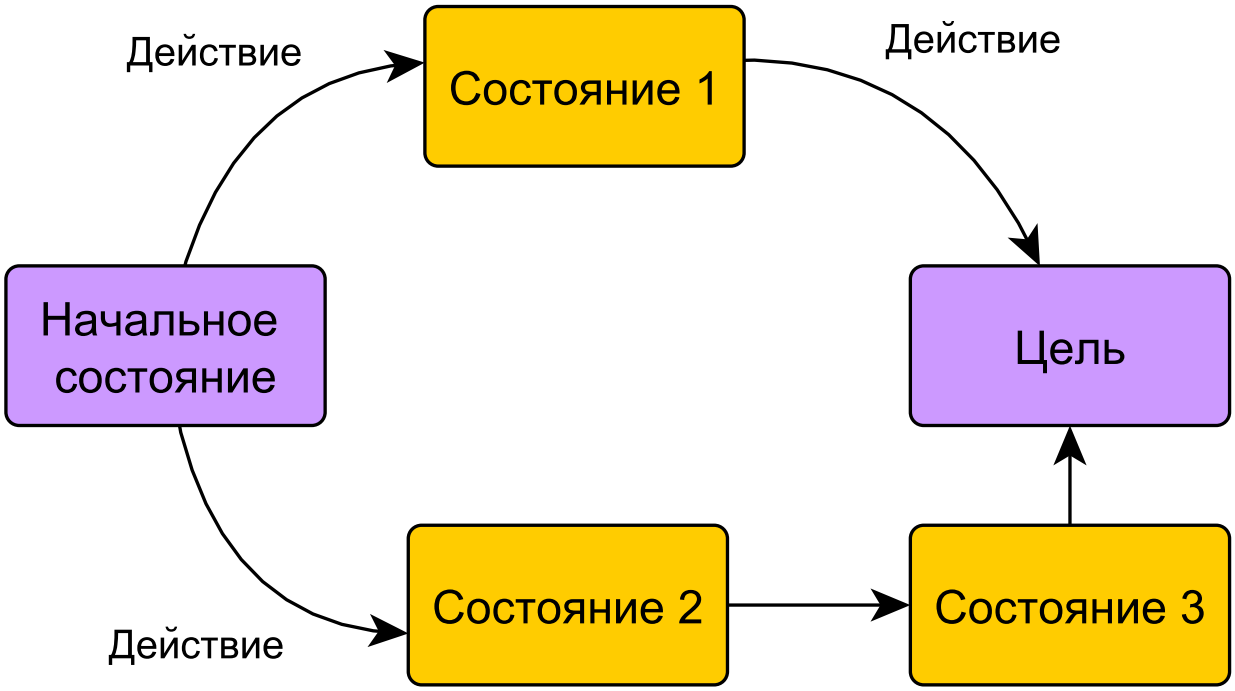

Рисунок 1.4 — Целе-ориентированная модель поведения

Целе-ориентированные модели являются крайне эффективным решением для достижения *тестируемости* и *полноты* программной системы. Построенная в результате процедуры логического вывода цепочка рассуждений может

быть проверена и неоднократно воспроизведена на одном и том же входном наборе данных.

Алгоритмы планирования и логического вывода появились достаточно давно и считаются классикой программирования и вычислительной техники. Целе-ориентированные модели управления начинают все чаще применяться в видео-играх, так как вычислительные мощности современных вычислительных систем стали достаточны для решения задач в режиме реального времени.

Для планирования действий может быть использован алгоритм GOAP [61; 62], который является упрощенной версией планировщика STRIPS (Stanford Research Institute Problem Solver) [63]. Эффективность целе-ориентированного ИИ на примере игры StarCraft наглядно продемонстрирована в [64].

В целом целе-ориентированные модели обладают высокой адаптивностью. Для каждого возможного действия агента известны условия, в которых действие в принципе может быть совершено, и последствия, к которым оно приводит. используя эти знания агент может использовать процедуру логического вывода для решения решения достаточно широкого класса задач. Однако, одной способности решать задачу в общем виде для достижения адаптивности недостаточно, необходимо, чтобы агент принимал решения за такое время, чтобы состояние окружающей среды не изменилось настолько, чтобы сделать полученное решение бесполезным.

Вычислительная сложность планирующих алгоритмов имеет комбинаторную природу. Предположим, что агент должен достичь множество целей Q в любой последовательности и существует M параметров системы $c_i, 1 <= i <= M$. Тогда количество состояний, в котором может находиться система будет равно произведению мощностей множеств значений этих параметров: $N_s = \prod_{i=1}^{M} |C_i|$. В одном из самых простых случаев, когда мощность множества значений любого параметра равна 1, задача поиска оптимального маршрута достижения всех целей практически эквивалентна классической задаче коммивояжера. Приведенная оценка носит исключительно качественный характер, но позволяет понять ограничение на применимость целе-ориентированных моделей в системах реального времени.

В 70-х годах были предложены алгоритмы планирования иерархических сетевых задач (Hierarchical Network Task planning(HTN))[65; 66]. HTN-планирование интересно тем, что в отличии от классических алгоритмов де-

композирует задачу на иерархические задачи так, что внутри каждой задачи пространство для поиска существенно уменьшается. HTN-планировщики эффективны для моделирования стратегического ИИ [67].

Помимо высокой вычислительной сложности целе-ориентированные модели обладают крайне существенным недостатком: формальная задача агента может не иметь решения вовсе. Это означает, что ресурсы, задействованнае на дорогостоящую операцию планирования будут потрачены впустую, а агент должен построить запасной план, который будет использоваться до тех пор пока получение точного решения не станет возможным. Таким образом, вместо реализации одной модели поведения необходимо реализовать несколько схожих таким образом, что в большинстве прикладных задач, как мы уже показывали на примере планирования маршрутов, можно безболезненно отказаться от наиболее сложной и дорогой из реализаций. Эта проблема является одной из причин, почему разработчики видео-игр отказываются от использования GOAP и ему подобных алгоритмов [68].

Интересный подход с генерацией сценария поведения NPC на основе построенного на этапе разработки HTN-плана представлен в работе [69]. Недостатком такого подхода является статичность алгоритма поведения NPC.

Иерархическое планирование может быть использовано в комбинации с деревьями поведения [70]. На верхнем уровне сценарий поведения агента определяется деревом поведения, в то время как решение текущих задач, представленных узлами дерева перекладывается на алгоритм поиска в ограниченном пространстве состояний. Частичным решением решением проблем построения точного плана можно считать использование стохастических алгоритмов, таких как метод Монте-Карло[71].

Таким образом целе-ориентированные модели поведения обладают высоким потенциалом, благодаря своей способности к решению задач в обобщенной форме. Однако, для эффективного применения в системах реального времени алгоритмы планирования дополняются большим количеством проблемно-ориентированных эвристик, необходимых для снижения времени отклика системы.

1.8 Численные модели агентов

Под *численными моделями агентов* понимают достаточно широкий класс моделей представляющих знания об окружающей среде в явной форме и использующих численные методы решения задач оптимизации при принятии решений.

Численные модели базируются на аналитических методах решения задачи. Наиболее часто численное моделирование используется при решении задачи перемещения агента в динамической среде. Задача может решаться с использованием методов линейного программирования, линейной алгебры и аналитической геометрии.

Одним из эффективных методов является метод виртуальных потенциальных полей [72—76]. Метод может быть использован как для решения задачи навигации, так и для принятия решений в фазовом пространстве. Точки фазового пространства, соответствующие желаемым состояниям системы являются аттракторами, а нежелательным – репеллерами. Точки фазового пространства, находящиеся ближе к аттракторам имеют больший потенциал. При использовании метода каждой точке пространства состояний ставится в соответствие некоторое численное значение. Агент стремиться попасть в точку поля с наибольшим значением используя градиент функции поля. Проблемой метода потенциальных полей является наличие локальных экстремумов.

К численным моделям также можно отнести системы так называемого роевые оптимизационные методы искусственного интеллекта [77; 78]. Методы роевой оптимизации, такие как муравьиный алгоритм, помогают эффективно находить приближенные решения задач на графах. Эффективность методов проявляется при построении систем с большим количеством агентов, например при моделировании процесса разработки программного обеспечения с открытым исходным кодом [79]. Каждый агент сам по себе представляет очень простую сущность, обладающую минимальным набором знаний об окружающей среде. В процессе моделирования принимает участие значительное количество агентов, что позволяет добиться системного эффекта – итоговое поведение роя оказывается существенно сложнее поведения индивидуальных агентов. Однако,

использование роевого интеллекта в системах с «умными» агентами не получило распространения.

Численные модели позволяют добиться компромисса между высокой производительностью и точностью получаемого решения за счет варьирования параметров ошибки, критериев оптимизации, количества итераций поиска и количества агентов, участвующих в моделировании. Исключение составляют модели, которые сводятся к решению сложных систем дифференциальных уравнений.

Важным недостатком нечетких моделей является непредсказуемость принимаемых агентом решений для разработчика. Оценка наиболее вероятного сценария поведения агента при известных параметрах нечеткой модели является нетривиальной вычислительной операцией для человека. Аналитическая модель предметной области, с которой оперирует агент, как правило имеет значительное количество параметров, систем ограничений и особых случаев, что весьма негативно сказывается на сопровождаемости и мобильности агентов.

1.9 Нейросетевые модели агентов

После определенного периода разочарования в нейросетвых моделях произошло их «повторное рождение». В последние годы нейронные сети активно применяются в разработке некоммерческого игрового ИИ в рамках различных исследовательских проектов [80—82]. К успешным примером использования нейронных сетей безусловно следует отнести AlphaGo, реализованный с использованием алгоритма, комбинирующего глубинное обучение и поиск Монте-Карло в дереве [83].

Нейросетевые модели активно используются в разработке креативных интеллектуальных систем. Такие системы могут заниматься, например, машинным творчеством[84]. В основе креативных систем обычно используются алгоритмы обучения с подкреплением[85].

Существуют примеры успешного применения нейронных сетей для построения моделей поведения[86]. Нейросетевые модели создают определенные сложности при использовании их в коммерческих проектов в качестве базы

для игрового ИИ. Основной их проблемой является непредсказуемость, в двух схожих ситуациях поведение агента построенного на нейронной сети может существенно различаться. Помимо проблемы воспроизводимости, существуют ситуации, в которых поведение интеллектуального агента должно быть строго регламентировано. В видео-играх, например, элементы ИИ используются для создания нарратива, и поведение некоторых агентов предопределено заранее. Нейронные сети классифицируют ситуации с определенной погрешностью, и любая ключевая ситуация может быть неверно распознана. Поиск и устранение подобных ошибок может быть весьма затруднен. Однако, существуют примеры успешного применения нейронных сетей в коммерческих проектах, и наличествуют тенденция к расширению области применения нейросетевых моделей [87]

Преимуществом нейросетевых моделей является высокая адаптивность и способность к самоорганизации [88]. Можно сказать, что нейросетевые модели хорошо зарекомендовали как сильные интеллектуальные методы, подходящие для задач с высокой степенью неопределенности, которые, как правило, не могут быть за приемлемое время решены классическими методами. В случае, когда от агента требуется следование заранее определенным сценариям, использование нейронных сетей становится вызовом для разработчика и создает проблемы, связанные с верификацией поведения агента.

1.10 Выводы

В первой главе даны определения понятиям агент и мультиагентная система. Рассмотрены классификационные признаки среды функционирования агентов и выделен актуальный класс МАС.

На основе критериев качества программного обеспечения, в соответствии со стандартами ISO и ГОСТ, сформулированы критерии качества МАС. Качество мультиагентной системы в значительной степени определяется качеством модели поведения агента, для оценки которого выбраны следующие критерии:

1. функциональность – способность агента рационально выполнять поставленные задачи;

2. адаптивность – способность агента корректировать свое поведение в соответствии с изменениями окружающей среды;
3. производительность – соотношение между количеством решаемых задач и затрачиваемыми вычислительных ресурсами;
4. сопровождаемость – характеристика модели соответствующая стоимости внесения изменений в поведение агента на этапе разработки.

Для рассматриваемого класса МАС проанализированы типовые задачи, существующие методы их решения и актуальные проблемы. Для обеспечения эффективного функционирования агента необходимо решить задачи навигации, принятия решений и адаптации.

В ходе анализа были исследованы следующие классы моделей поведения агентов: конечно-автоматные, целе-ориентированные, численные и нейросетевые. В качестве критериев оценки качества моделей были использованы адаптированные стандартные критерии оценки качества программного обеспечения: сопровождаемость, производительность и адаптивность. Выбранные критерии носят качественный характер.

Сопровождаемость модели определяется цикломатической сложностью ее реализации. Так конечно-автоматная модель требует реализации всех возможных условных ветвлений, целе-ориентированная – только правил выполнения логическиого вывода.

Производительность модели поведения агента может быть оценена временем, которое необходимо модели на решение задачи. Время зависит от количества и сложности операций по нахождению решения.

Адаптивность модели поведения агента определяет способность агента рационально выполнять поставленные задачи при изменении состояния окружающей среды.

Результаты сравнительного анализа МАС, основанных на рассматриваемых моделях поведения агентов представлены в таблице 1.

Таблица 1 — Сравнительная таблица МАС

Модель поведения агента	Сопровождаемость	Производительность	Адаптивность
Конечно-автоматная	Средняя высокая цикломатическая сложность	Высокая $O(N_{nodes})$	Низкая фиксированное поведение
Целе-ориентированная	Высокая низкая цикломатическая сложность	Низкая $O(N_{states}!)$	Высокая логический вывод
Численная	Низкая высокая цикломатическая сложность	Средняя «тяжелые» вычисления	Средняя аппроксимация
Искусственная нейронная сеть	Низкая наличие неявных зависимостей	Высокая $\approx O(N^2_{neurons})$	Высокая обучение

В ходе анализа было выявлено, что рассмотренные модели поведения обладают низким значением как минимум одного параметра. Качественное решение задач агентом требует высоких вычислительных затрат. Оптимизация МАС с точки зрения производительности приводит к сильной связанности компонентов системы и снижению ее сопровождаемости. Таким образом, важной задачей при разработке МАС является достижение баланса между адаптивностью, производительностью и сопровождаемостью программной реализации МАС. Для повышения качества модели поведения агента необходимо

– сократить количество вычислительных ресурсов, затрачиваемых на принятие решений;
– обеспечить модульность архитектуры агента;
– уменьшить количество зависимостей между модулями агента;

Точное решение задач агентов требует использования значительных вычислительных затрат: процессорного времени и оперативной памяти, что неприемлемо в системах реального времени, в которых необходим быстрый отклик.

Динамические изменения окружающей среды требуют от агента высокой адаптивности. В большинстве случаев невозможно заранее, на этапе разработки, предусмотреть все возможные состояния окружающей среды, поэтому необходимо, чтобы агент в случае возникновения непредвиденной ситуации мог самостоятельно сгенерировать рациональное решение.

В жизненном цикле программных агентов неизбежно обнаружении ошибок и изменение требований к системе. Архитектура агента должна обеспечи-

вать возможность внесения изменений, направленных на реализацию нового функционала и исправление ошибок. Проблемно-ориентированность интеллектуальных агентов и высокая связанность программного кода существенно увеличивает стоимость поддержки.

Глава 2. Модель агента на основе расширенного дерева поведения

2.1 Формальная модель мультиагентной системы

Взаимодействие агента с окружающей средой является определяющим фактором мультиагентной системы. В соответствии с [89; 90] МАС должна быть определена на трех уровнях: прикладной логики, платформы выполнения и физической инфраструктуры.

На уровне прикладной логики агенты осуществляют операции, непосредственно связанные с решением задачи. Уровень платформы выполнения, или промежуточный уровень (Middleware), является интерфейсом между аппаратной платформой, который «склеивает» распределенные компоненты системы. Уровень физической инфраструктуры связан с объектами реального мира, такими как аппаратное обеспечение ЭВМ. Физическая инфраструктура должна приниматься во внимание даже при разработке виртуальных МАС, таких как видеоигры и симуляторы, так как от этого зависит эффективность использования вычислительных ресурсов платформы.

Мультиагентная система определяется множеством элементов системы и связей между ними. Основными элементами мультиагентной системы являются агенты, объекты окружающей среды и события которые в ней происходят. Элементы МАС связаны между собой законами функционирования системы – реакцией окружающей среды на действие или бездействие агентов[91]. В общем виде модель мультиагентной системы Ψ может быть представлена кортежем

$$\Psi = \langle \Theta, P_\Psi, E, \delta_\Psi, S_\Psi, s_{\Psi 0} \rangle \tag{2.1}$$

где:

Θ – множество объектов окружающей среды;
P_Ψ – множество параметров окружающей среды;
E – множество событий, которые могут произойти в среде;
S_Ψ – множество всех возможных состояний системы;
$s_{\Psi 0} \in S_\Psi$ – начальное состояние системы;
δ_Ψ – функция перехода между состояниями S_Ψ.

В каждый момент времени окружающая среда характеризуется ее состоянием. Состояние окружающей среды $s_\Psi(t)$ в момент времени t формируется

состоянием всех объектов и значениями всех параметров среды. Множество S_Ψ всех возможных состояний мультиагентной системы является декартовым произведением

$$S_\Psi = S_{\theta 1} \times \ldots \times S_{\theta K} \times V_{p1} \times \ldots \times V_{pM} \times E \tag{2.2}$$

где:

K – количество объектов окружающей среды;
M – количество параметров среды;
$S_{\theta i}$ – множество состояний объекта $\theta_i \in \Theta | i \in [1; K]$;
V_k – множество значений параметра $p_k \in P_\Psi | k \in [1; M]$.

Все объекты МАС могут быть классифицированы по своей роли в системе на статические и интерактивные. Статические объекты не меняют своего состояния в процессе моделирования. С помощью статических объектов осуществляется моделирование относительно неизменных объектов реального мира – элементов ландшафта, стен, деревьев. Интерактивные объекты обладают собственной логикой изменения состояния и активно реагируют на действия агента. К таким объектам относятся переключатели, двери, контейнеры. Отдельным классом интерактивных объектов являются предметы $\zeta \in Z \subset \Theta$. Предмет может находиться в свободном состоянии в пространстве окружающей среды либо помещен «внутрь» другого объекта – контейнера. Агенты могут подбирать, переносить и использовать предметы в качестве инструментов для воздействия с окружающей средой.

Модель времени в мультиагентной системе является последовательной. Непрерывный интервал моделирования $[0; T]$ разбивается на примерно-равные промежутки. Ключевые моменты времени, по которым осуществляется разбиение, составляют множество ΔT.

Изменение состояния мультиагентной системы $\Delta s_{\Psi t}$ определяется событием $e(t) \in E$ произошедшим в момент времени t и текущим состоянием системы. Считается, что агенты имеют полные знания о законах функционирования системы, которые определяются функцией

$$\delta_\Psi : E \times \Delta T \to \Delta S_\Psi \tag{2.3}$$

В целях упрощения модели будем считать, что изменения окружающей среды обладают свойством суперпозиции и ассоциативности. Если два или более событий происходят единовременно, то итоговое изменение окружающей среды является суммой изменений от каждого события. Предполагается, что порядок обработки событий, произошедших в пределах одной итерации, не играет роли.

Множество агентов рассматриваются как подмножество интерактивных объектов и часть окружающей среды [92], то есть $ainA \subset \Theta$. Модель агента может быть представлена следующим кортежем:

$$a = \langle P_A, \Omega, M, H, \beta \rangle \tag{2.4}$$

где:

P_A – множество параметров агента;
Ω – множество сенсоров;
M – память агента;
H – множество приводов;
β – функция поведения агента.

Сенсоры агента отвечают за восприятие событий, приводы – за перемещение в пространстве, взаимодействие с интерактивными объектами и коммуникацию с другими агентами. Результаты восприятия после обработки сохраняются в памяти агента.

Агент может взаимодействовать с другим агентами и *интерактивными объектами* совершая над ними *действия*. Для каждого интерактивного объекта известны физические аспекты его модели и поведенческие реакции при различных способах взаимодействия. При совершении действий агент может использовать *инструменты* – предметы которые агент «носит» с собой. Взаимодействие происходит на дистанции, определяемой способом взаимодействия – *классом действия* и используемыми инструментами. В каждый момент времени привод агента $\eta \in H$ может совершать не более одного действия ξ. Длительность действия также определяется классом действия и инструментами.

Последовательность действий, выполняемых агентом в процессе моделирования формирует поведение агента [93], которое проявляется и как непосредственно наблюдаемые действия, и как изменение скрытого внутреннего состояния – памяти. Поведение агента может быть представлено функцией β отобра-

жающей состояние окружающей среды в действие агента и изменение состояния памяти.

$$\beta : S_\Psi \times S_M \times E \times \Delta T \to \Delta_{S_H} \times \Delta_{S_M} \tag{2.5}$$

где:

Δ_{S_M} – множество изменений состояния памяти агента;
Δ_{S_H} – множество действий агента.

Каждое действие агента $\xi \in \Xi$ может быть описано кортежем

$$\xi = \langle \eta, \theta_{target}, \zeta_{tool}, \delta_\xi \rangle \tag{2.6}$$

где:

$\eta \in H$ – привод, который задействуется агентом;
$\theta_{target} \in \Theta$ – целевой объект, над которым производятся манипуляции;
$\zeta_{tool} \in Z_A$ – инструмент или ресурс, используемый агентом при взаимодействии;
δ_ξ – функция воздействия.

Функция воздействия δ_ξ описывает изменение состояния окружающей среды в процессе выполнения действия агентом.

$$\delta_\xi : S_\Psi \times \Theta \times Z \times \Delta T \to \Delta S_\Psi \tag{2.7}$$

Каждое агент воздействует на окружающую среду независимо от других. Результирующее изменение окружающей среды $\Delta s_\Psi(t)$ в момент времени t является суммой воздействий всех агентов:

$$\Delta s_\Psi(t) = \sum_{i=1}^{N} \delta_{\xi_i}(s_\Psi(t), \theta_{target_i}, \zeta_{tool_i}, t) \tag{2.8}$$

С точки зрения теории полезности эффективность агента оценивается как суммарная полезность, полученная за все время моделирования T. Каждое действие ξ, совершаемое агентом вносит некоторый вклад в суммарную полезность. Вклад действия определяется не только свойствами самого действия, но и моментом его выполнения.

$$Y(T) = \int_0^T U(t)dt \tag{2.9}$$

Таким образом, задача разработки мультиагентных систем сводится к формированию функции поведения агента β таким образом, чтобы полученная в результате последовательность действий, максимизировала бы суммарную полезность Y.

2.2 Концепция модели поведения агента

Предлагаемая модель агента включает в себя взаимодействующие посредством интерфейсов типовые компоненты, наиболее часто используемые при проектировании МАС. К ним относятся сенсоры, приводы, память, в которой хранится информация, ассоциированная с объектами виртуального мира, модуль принятия решений и модуль навигации. Предлагаемая структура агента представлена рисунке 2.1.

Сенсоры агента являются источником информации об окружающей среде. Сенсор осуществляет запрос данных о свойствах наблюдаемой группы объектов. Собранные данные кэшируются в кратковременную память. Для каждого объекта который попадает в область восприятия сенсора или покидает ее генерируется соответствующее событие.

Память агента в течении длительного времени хранит обработанные результаты восприятия, которые включают в себя информацию о произошедших событиях и свойствах объектов окружающей среды. Память агента устроена по принципу классной доски (blackboard). Переменные в памяти имеют уникальные идентификаторы и доступны всем компонентам агента. Особенностью предложенной концепции являются ассоциативные переменные, значения которых связаны с объектами окружающей среды. Каждая переменная имеет свое время жизни, которое определяет актуальность записанной информации. Все переменные являются изменяемыми.

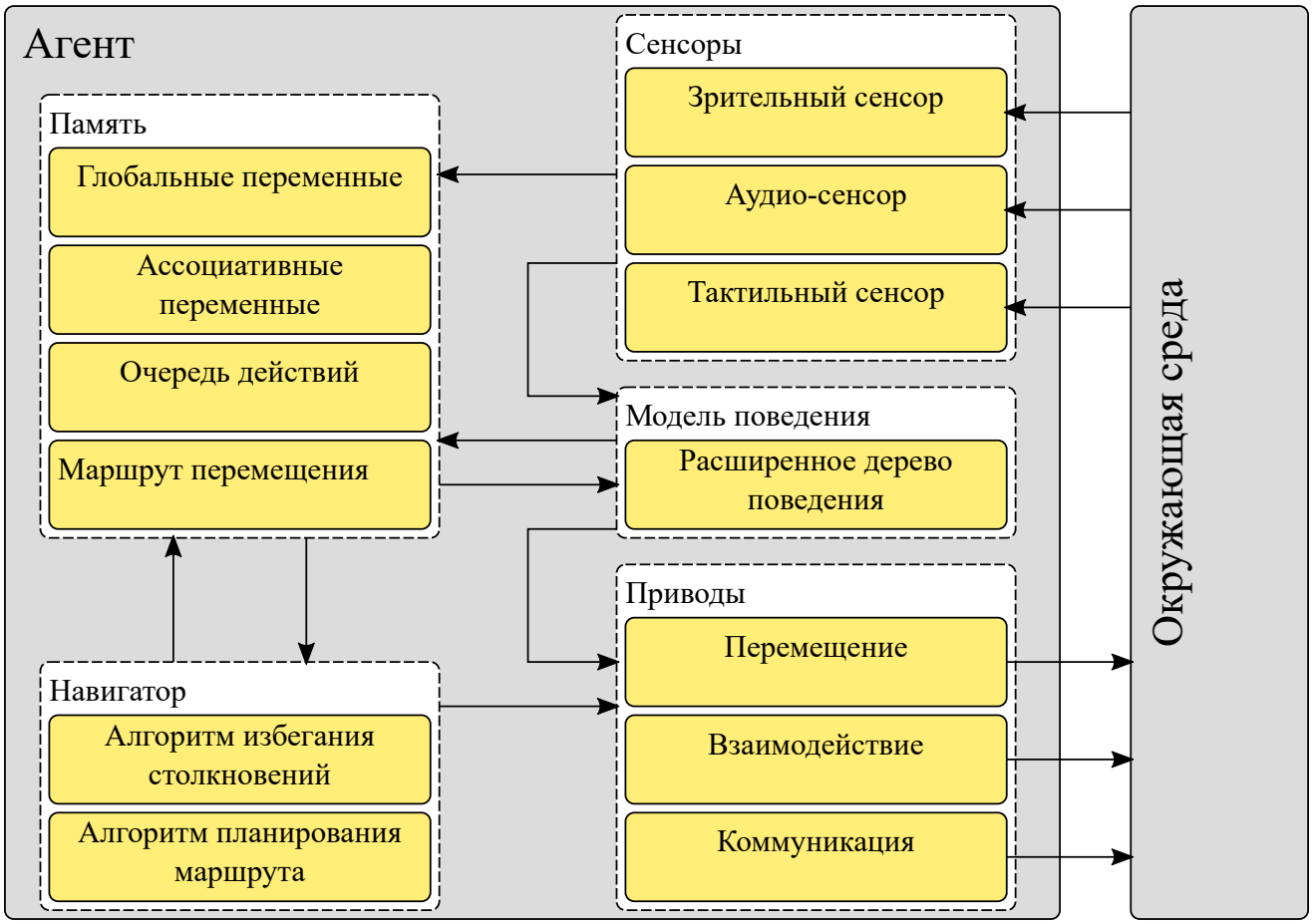

Рисунок 2.1 — Предлагаемая схема агента

Приводы осуществляют действия агента, такие как перемещения(локомоции) и взаимодействие с интерактивными объектами. Состояние приводов может быть визуализировано с использованием анимации.

Навигатор осуществляет планирование маршрута и выбор направления движения с учетом возникающих на пути препятствий. Входными данными для навигатора является некоторая область пространства относительно указанного объекта окружающей среды, к которому или от которого агент желает совершить перемещение. Построенный маршрут сохраняется в памяти для дальнейшего использования. В случае, если агент уклоняется от маршрута или целевая область значительно меняется навигатор рассчитывает маршрут повторно.

Модуль принятия решений реализует функцию агента – преобразует входы в виде восприятий сенсоров и переменных в памяти в последовательность выполняемых действий. Для реализации модуля принятия решений предлагается использовать адаптивное дерево поведения, изменяющее свою структуру в соответствии с текущими задачами агента. Узлы дерева поведения осуществляют контроль над выполнением действий и достижением целей агента.

С позиции утилитарного подхода агент в процессе функционирования стремится максимизировать суммарную полезность – скалярную величину, позволяющую количественно оценить уровень «благополучия» агента. За время моделирования T_{sim} суммарная полезность Y, полученная агентом может быть представлена выражением

$$Y = \int_0^{T_{sim}} U(t)dt \tag{2.10}$$

где

T_{sim} – время моделирования;
$U : S_\Psi \to \mathbb{R}$ – функция добавочной полезности.

Дальнейшее описание задачи в общем виде приводят к системам дифференциальных уравнений и численным методам их решения [94]. При решении прикладных задач численные методы не всегда приемлемы в условиях ограниченных вычислительных ресурсов.

Вместо непрерывной модели воспользуемся дискретным приближением. Предположим, что для достижения наибольшего значения суммарной полезности агент должен последовательно побывать в некоторых состояниях, которые соответствуют тактическим целям агента. Тогда целевая функция может быть апроксимирована суммой полезности, получаемой при достижении каждой из целей агента:

$$Y \simeq \sum_{i=0}^{n} u(q_i) | q_i \in Q \tag{2.11}$$

где

Q – множество всех возможных целей агента;
$u : Q \to \mathbb{R}$ – функция полезности цели;

Когда агент достигает состояния, которое соответствует цели q_i можно считать, что он увеличивает суммарную полезность на некоторое значение $u(q_i)$. Если агент находится в состоянии, которое не соответствует ни одной из целей, то добавочная полезность полагается равной нулю.

В каждый момент времени t для агента должно быть известно множество текущих целей $Q_t \subset Q$, из которых агент должен выбрать одну активную цель с наибольшим приоритетом.

Приоритет цели может быть выражен функцией от предполагаемой добавочной полезности и затрат на ее достижение. В основе предлагаемой архитектуры лежит предположение, что агент должен отдавать предпочтения целям, которые имеют наибольшую добавочную полезность при наименьших затратах ресурсов и времени. Применяемая «жадная» эвристика не учитывает зависимости между целями и долгосрочные последствия от достижения цели, но часто позволяет получить эффективную модель поведения в краткосрочной и среднесрочной перспективе.

Отсутствие зависимостей между целями позволяет снизить количество параметров задачи и, как следствие, повысить сопровождаемость и производительность МАС в целом. Каждая цель агента ассоциируется с одним единственным объектом и описывает его желаемое состояние. Задача агента – выбрать наиболее приоритетную потенциально-достижимую цель и построить последовательность действий для ее достижения. В контексте каждой цели агент может использовать специализированные модели представления окружающей среды, алгоритмы поиска решений и проблемно-ориентированные эвристики.

2.3 Расширенное дерево поведение

Модуль принятия решений является наиболее важным компонентом агента. Для его реализации разработана модель расширенного дерева поведения(Extended Behavior Tree). Дерево поведение отвечает за мгновенные реакции агента на происходящие события, последовательное выполнение спланированных действий и контролирует достижение целей.

Классическое дерево поведения является статической конечно-автоматной моделью, к недостаткам которого относится существенное снижение сопровождаемости с ростом количества возможных действий агента. Для того, чтобы избавиться от присущих статическому дереву недостатков предлагается снизить общее количество элементов модели за счет замены узлов ветвления (селекто-

ров) на унифицированные узлы с приоритетом и удаления из дерева неиспользуемых узлов. Таким образом, предлагаемое расширение позволяет адаптировать свою структуру в соответствии с текущими целями агента.

Структура дерева представлена на рисунке 2.2. Каждую итерацию моделирования агент осуществляет обход дерева поведения в глубину, выполняя содержащиеся в узлах инструкции в соответствии с алгоритмом 1. Расширение включает в себя дополнительные классы узлов, осуществляющих адаптацию дерева к изменениям в окружающей среде.

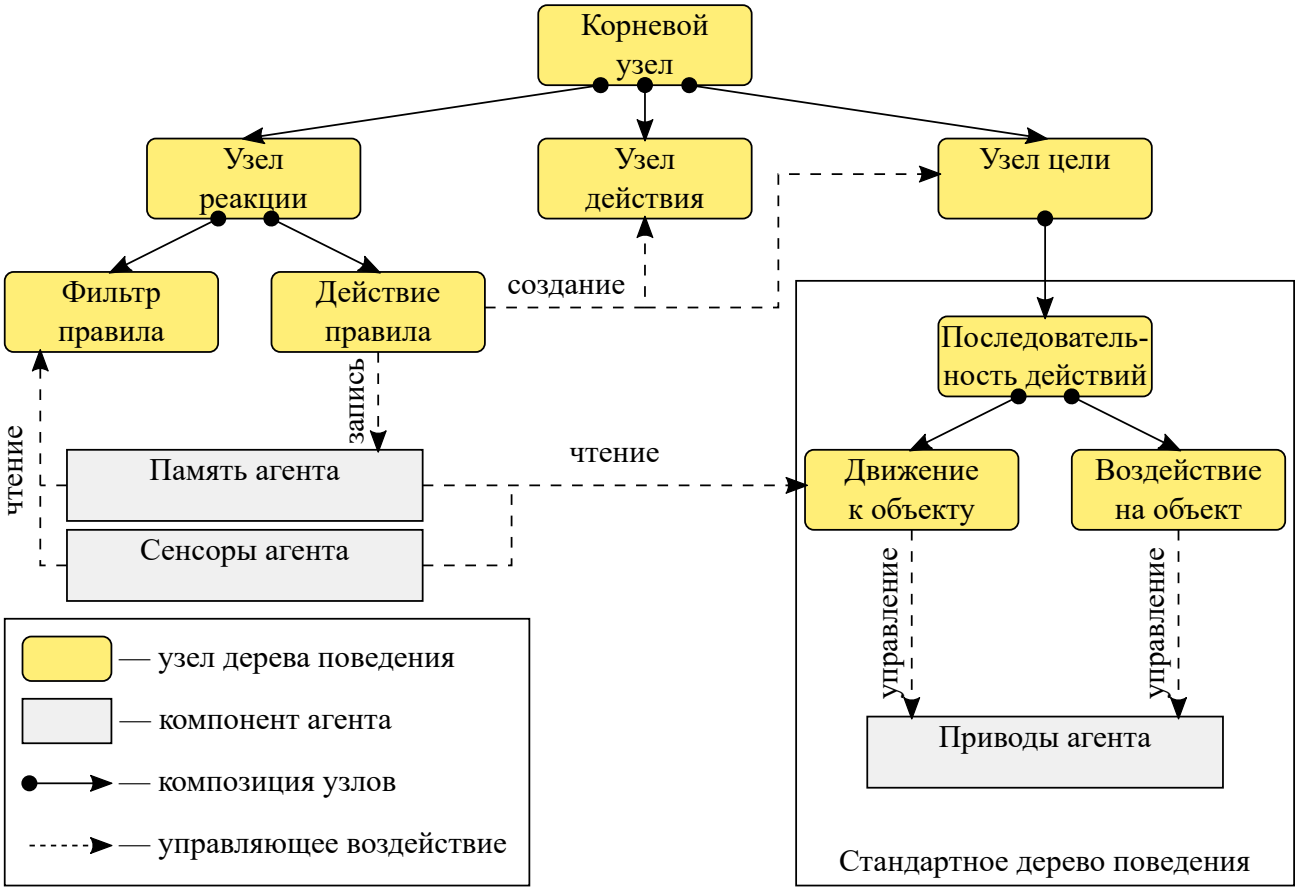

Рисунок 2.2 — Расширенное дерево поведения

Узел реакций отвечает за обработку событий, генерируемых сенсорами агента. Узел представляет собой продукционное правило, реализующее примитивную реакцию, эффект от которой наступает незамедлительно. Правило состоит из фильтра – предиката от события, и действия – инструкции по записи в память значения или изменения структуры дерева. Использование явнозаданных правил позволяет добиться от агента следования определенному сценарию тогда, когда это необходимо. С помощью этого механизма могут быть

реализованы контекстные анимации, эмоциональные реакции. Результатом действия правила так же может являться генерация новых целей агента.

Узел действия непосредственно моделирует взаимодействие агента с интерактивным объектом в окружающей среде.

С помощью *узлов целей* реализуется проактивное поведение агента. Каждый такой узел соответствует описывает желаемое состояние окружающей среды и некоторого целевого объекта. Метод адаптации дерева поведения построен на использовании теории полезности, аналогично [95], – порядок обхода узлов целей зависит от предельной полезности достижения цели. Чем больше ожидаемая полезность и меньше ожидаемые затраты, тем более высокий приоритет имеет соответствующий узел. Далеко не все цели могут быть достижимы из текущего состояния агента, в этом случае приоритет узла будет равен нулю. Если цель достигнута или ее достижение становится принципиально невозможным, то она становится неактуальной. Узлы, соответствующие неактуальным целям удаляются из дерева.

Алгоритм 1 — Выполнение расширенного дерева поведения

procedure ExecuteTree($root$,t)
  for $e_t \in E_t$ **do**
    HandleEvent(e_t) ▷ Обработать наблюдаемые события
  end for
  for $q \in Q_t$ **do** ▷ Пересчитать приоритеты целей
    $q.IsActual \leftarrow c_i(s_\Psi, s_A, e_t)$
    $q.Priority \leftarrow P_i(s_\Psi, s_A, e_t)$
  end for
  $Q_t \leftarrow \{q | q \in Q_t, q.IsActual\}$ ▷ Удалить неактуальные цели
  $q_{current} = \text{argmax}_{q \in Q_t}\, q.Priority$ ▷ Выбрать приоритетную цель
  return ExecuteTree($q_{current}, t$) ▷ Передача управления узлу цели
end procedure

Таким образом, в каждый момент времени агент может выполнять одно действие в процессе достижения наиболее приоритетной цели, выбор которой осуществляется в корневом узле. Для того, чтобы уменьшить количество вычислений, обработка целей проводится в два этапа. На этапе предобработки отсекаются цели, достижение которых в силу различных факторов в текущий момент невозможно. Оставшиеся упорядочиваются в порядке убывания приори-

тета. На втором этапе агент последовательно обходит поддерево, соответствующее активной цели, в соответствии с моделью классического дерева поведения.

2.4 Модель цели агента

Цели агента всегда описывают желаемое состоянием некоторого *ассоциированного объекта*, которым может быть любой агент или интерактивный объект окружающей среды. В поддереве узла цели осуществляется обработка событий, связанных с ассоциированным объектом. События сигнализируют об изменении состояния наблюдаемого объекта, которое может привести к изменению приоритета или актуальности цели. Цель считается актуальной, если состояние ассоциированного объекта еще не соответствует условиям цели и хотя бы одно из целевых состояний является достижимым.

Модель цели агента представлена кортежем

$$q_i = \langle \theta_i, P_i, c_i, A_i \rangle | q_i \in Q \tag{2.12}$$

где:

Q – множество всех актуальных целей агента;
O_i – объект окружающей среды, с которым ассоциирована цель;
P_i – функция оценки приоритета цели;
c_i – предикат определяющий соответствие состояния s целевому состоянию;
A_i – поддерево, реализующее алгоритм достижения цели q_i.

Предложенная модель позволяет разбить дерево поведения на поддеревья таким образом, чтобы последовательное изменение порядка обхода, а также включение и исключение поддеревьев позволяло добиться рационального поведения агента. Каждый узел цели становится слабо связанным модулем, предоставляющим интерфейс для высокоуровневого управления целями в корневом узле.

Изменение множества целей происходит исключительно в виде реакции на события в окружающей среде. Генерация новых целей осуществляется правилом в узле реакции, а удаление цели q_i – корневым узлом, в соответствии с предикатом c_i.

2.5 Метод оценки приоритетов целей агента

Оценка приоритетов целей является одной из критических точек системы. С одной стороны функция оценки должна быть максимально точной для достижения эффективности действий агента, с другой – ее вычисление должно требовать минимально возможное количество ресурсов.

Зная порядок обхода целей и используя наиболее подходящие алгоритмы для их достижения, агент может эффективно функционировать в статической среде. Однако, в динамической среде приоритет цели может измениться в процессе функционирования агента. Задача выбора оптимального порядка целей обхода близка к классической задаче о коммивояжере, так как для точного определения приоритета одной цели необходимо знать последствия всех целей агента. В случае неравенства классов $N \neq NP$ эта задача имеет сложность порядка $O(N) = 2^N$. Для систем реального времени такая сложность является неприемлемой. Для решения этой проблемы воспользуемся эвристикой о том, что агенту выгоднее двигаться к ближайшим целям, достижение которых наиболее вероятно, и которые в среднесрочной перспективе вносят больший вклад в суммарную полезность.

Для решения проблемы оценки приоритетов целей был разработан набор эвристик, позволяющий оценивать приоритеты целей независимо друг от друга. Приоритет цели – это комплексная количественная оценка, которая вычисляется на основе некоторых наблюдаемых агентом свойств цели – *критериев приоритета*. Эмпирическим способом было подобрано множество значимых в ограничениях задачи критериев, которое включает в себя:

– расстояние до точки навигации, d;
– ожидаемые затраты ресурсов, c;
– предельная полезность достижения цели, u;
– вероятность успешного достижения цели, p.

В процессе выполнения значения критериев приоритета для каждой цели будут меняться, что приведет к тому, что изменится и порядок их обхода целей агентом. При отсутствии ощутимых внешних воздействий приоритет текущей цели будет только возрастать. Для ее достижения будет требоваться все меньше ресурсов. Вероятность успешного завершения повысится, так как

будет оставаться меньше возможностей для нежелательных событий. Расстояние до точки навигации не включено в затраты ресурсов, так как для оценки расстояния уже существует множество эвристических функций, позволяющих добиться хороших результатов при планировании маршрута.

Значение каждого критерия должно быть нормализовано до диапазона значений $[0; 1]$. Полученная многокритериальная оценка может быть свернута в номинальную оценку приоритета.

В качестве агрегирующей функции оценки рассматривалась аддитивная взвешенная свертка

$$P_q(S) = \sum_{i=1}^{N} k_i \cdot f_i(S) \tag{2.13}$$

и мультипликативная свертка

$$P_q(S) = \prod_{i=1}^{N} k_i \cdot f_i(S) \tag{2.14}$$

Поведение агента в случае использовании аддитивной свертки будет 2.13 более устойчивым, но в тоже время агент станет хуже реагировать на приближение значений параметров к граничным значениям, поэтому в качестве итоговой свертки выбрана мультипликативная 2.14. Такая функция гораздо лучше отражает нелинейный характер зависимости между различными критериями. В случае приближения одного из параметров к нулю приоритет цели будет существенно падать, что не позволит агенту совершать малоэффективные, дорогие или трудно выполнимые действия. Таким образом, результирующая функция оценки приоритета принимает вид:

$$P_q(s) = d_q(s) \cdot c_q(s) \cdot u_q(s) \cdot p_q(s) \tag{2.15}$$

Иногда для улучшения результата агент должен некоторое время продолжать попытки достижения текущей цели, даже если ее приоритет становится незначительно меньше, чем у другой цели. Для решения этой задачи было предложено использовать пороговое значение приоритета. В процессе упорядочивания целей, цель имевшая наивысший приоритет на предыдущем шаге итерации будет оставаться на первой позиции, до тех пор, пока не найдется цель, приоритет которой превысит приоритет текущей цели на относительной

пороговое значение, по условию $P_{new}(s) > P_{current}(s)(1+\theta)$. Варьируя значение θ можно добиться большей или меньшей чувствительности агента к изменениям окружающей среды.

Рассмотрим более подробно каждый критерий, используемый в функции оценки приоритета.

По аналогии с методом потенциальных полей значение *критерий дистанции* убывает по мере удаления агента от объекта, с которым ассоциирована цель. Качественный характер зависимости приоритета цели от дистанции до точки навигации показан на рисунке 2.3. При использовании метода потенциальных полей влияние аттракторов и репеллеров на агента часто описывают обратно-квадратичной функцией, аналогичной функции описывающей гравитационное взаимодействие. Схожая функция использована для качественного описания зависимости между расстоянием до ассоциированного объекта и приоритетом цели.

$$d_q(s) = \frac{1}{(?a \cdot |\overrightarrow{G_s} - \overrightarrow{A_s}| + 1)^{k_d}}, a > 0, k_d > 1 \tag{2.16}$$

где

$\overrightarrow{G}$ – координаты точки навигации;
$\overrightarrow{A}$ – координаты агента;
a, k_d – управляющие коэффициенты.

Управляющие коэффициенты a и k_d эмпирически подбираются таким образом, чтобы обеспечивать более рациональное принятие решений агентом. Помимо обратно-квадратичной может применяться также обратная или обратно-степенная зависимость. Обратно-степенные функции с более высоким показателем степени позволяют получить более стабильное поведение агента. В ходе экспериментов было установлено, что оптимальными значениями показателя степени являются значения из диапазона [1;5]. Чем выше показатель степени k_d, тем ближе к началу координат находится точка перегиба, и тем меньше радиус в котором агент может эффективно упорядочивать цели руководствуясь критерием дистанции. Управляющий коэффициент a также изменить размеры области, в которой цель начинает получать более высокий приоритет.

Фактор затрачиваемых ресурсов вычисляется исходя из оценочной стоимости, которая назначается всем предметам-ресурсам. Чем больше ресурсов

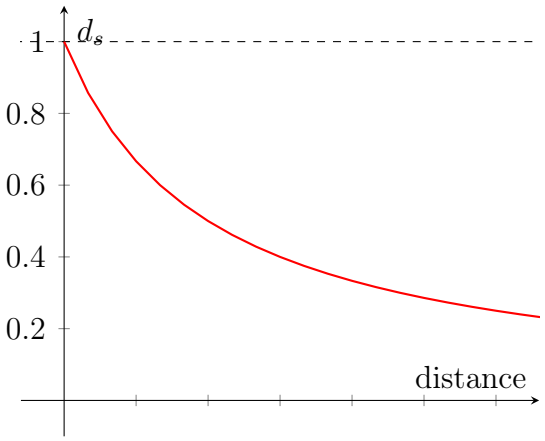

Рисунок 2.3 — График зависимости приоритета цели от дистанции

необходимо затратить для выполнения цели, тем меньше ее приоритет. Качественный характер зависимости показан на рисунке 2.4. Для оценки значения критерия ресурсов будем использовать формулу

$$c_q(s) = (1 - \frac{r_{spent}}{r_{total}})^{k_c}, k > 0 \tag{2.17}$$

где:

r_{spent} – стоимость затрачиваемых ресурсов;
r_{total} – суммарная стоимость имеющихся ресурсов;
k_c – управляющий коэффициент.

Если стоимость всех имеющихся ресурсов r_{total} известна агенту, то количество и тип затрачиваемых ресурсов для каждой цели должны определяться индивидуально. В разработанном прототипе для оценки используется приблизительная формула, которая предполагает, что агент будет использовать расходуемый ресурс только одного типа. В условиях сбалансировано-установленных цен на ресурсы стоимость использования ресурсов-заместителей не должна сильно отличаться. Значение управляющего коэффициента k_c позволяет регулировать «жадность» агента.

Фактор полезности достижения цели вычисляется индивидуально для каждого класса целей. Общим свойством для многих функций, описывающих данный критерий является соответствие закону убывающей предельной полезности, – добавочная полезность цели всегда положительна, но убывает по мере достижения агентом схожих целей. Так, например, если агент в игре собрал большое количество боеприпасов, то цель «подобрать предмет-боеприпасы» бу-

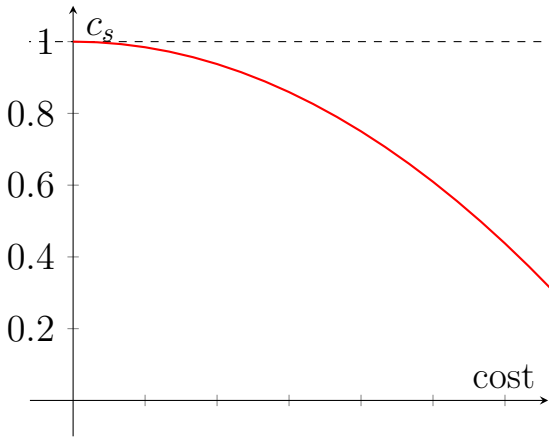

Рисунок 2.4 — График зависимости приоритета цели от затрачиваемых ресурсов

дет иметь меньший приоритет. Поскольку агент способен расходовать ресурсы, то соблюдение закона убывающей предельной полезности не является строгим.

Ценность выполнения цели не является постоянной величиной и может зависеть от многих свойств агента. Так ценность боеприпасов в игре зависит от наличия соответствующего оружия, аптечек – от текущего количества уровня здоровья. В разработанном прототипе значение критерия предельной полезности определяется эмпирически подобранной функцией

$$u(s) = \underset{[0;1]}{\text{clamp}}\left(1 + k_u \cdot (r_{total} + a_u), 0, 1\right), a_u < 0, k_u < 0 \tag{2.18}$$

где:

r_{total} – суммарное количество ресурса, которым владеет агент
a_u, k_u – управляющие коэффициенты

Вероятность достижения цели так же должна задаваться индивидуально для каждого класса целей. Наличие конкуренции и недетерминированности не позволяет точно оценить число благоприятных и неблагоприятных исходов. В прототипе используется эмпирическая кусочно-линейная функция, вычисляющая вероятность от расстояния до цели, количества конкурирующих агентов в окрестности и уровня здоровья агента.

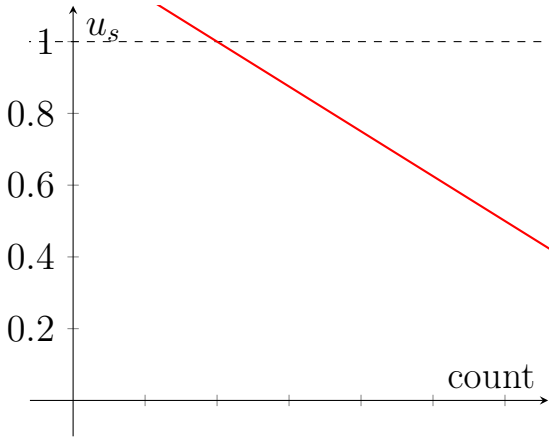

Рисунок 2.5 — График зависимости приоритета цели от предельной полезности

2.6 Память агента

Для того, чтобы эффективно выполнять задачи в долгосрочной перспективе, агенту необходимо запоминать произошедшие события и свойства окружающих объектов. Память агента устроена по принципу классной доски (blackboard). Обработанные результаты восприятия сохраняются в *памяти* агента в виде переменных, ассоциированных с объектами окружающей среды. Любой узел дерева поведения может прочитать или записать значение, хранящееся по указанному ключу.

Переменная в памяти агента описывается кортежем:

$$v_m = \langle id, \theta_a, v, T_{forget} \rangle | v_m \in V \tag{2.19}$$

где:

V – множество всех переменных;
id – уникальный идентификатор переменной;
$\theta_a \in \Theta$ – ассоциированный объект, может отсутствовать;
v – значение переменной;
T_{forget} – момент времени забывания.

Использование ассоциативных переменных позволяет частично структурировать знания агента. Вместо использования общей глобальной области видимости, агент локализует свои знания о каждом объекте окружающей среды. Переменные в глобальной области видимости не ассоциированы ни с одним объ-

ектом окружающей среды, для них $\theta_a = null$. Глобальные переменные агент может использовать для кэширования результатов запросов, а также для того, чтобы передавать параметры в узлы дерева поведения. Ассоциативные переменные помогают точнее оценить приоритет цели, связанной с наблюдаемым объектом. Таким образом, агент частично реализует фреймовую модель представления знаний.

Поскольку агент действует в динамической среде, факты в памяти агента не могут оставаться неизменными. Если наблюдаемые свойства объекта меняются, агент должен перезаписать значение соответствующей переменной. Важным критерием является частичная обозримость окружающей среды. Значения переменных, связанных с объектами, которые агент не может наблюдать в течении некоторого промежутка времени могут потерять актуальность. Для решения этой проблемы реализован механизм забывания на основе временных маркеров.

Каждый факт имеет свое, возможно бесконечное, время жизни. Временной маркер T_{forget} указывает на момент времени, в котором значение факта становится неактуальным. Временной маркер сдвигается вперед всякий, раз когда агент перезаписывает значение переменной.

Механизм временных маркеров позволяет эффективно реализовать алгоритмы исследования окружающей среды. Агент может создавать и размещать в пространстве *виртуальные объекты*, помечая значимые области пространства. Примерами виртуальных объектов являются маршрутные точки, точки возникновения шума. Виртуальный объект не может взаимодействовать с окружающей средой и существует исключительно в пределах памяти агента. В то же время агент может передать информацию о виртуальном объекте другому агенту в виде сообщения.

Множество ассоциативных переменных определяется динамически в процессе моделирования, а не на этапе компиляции, что позволяет снизить связность модели агента и повышает возможности повторного использования компонентов.

2.7 Метод динамического изменения структуры дерева поведения

Для эффективного функционирования агенту необходимо корректировать свое поведение в соответствии с изменением состояния окружающей среды. Агент воспринимает окружающую среду и ее изменения при помощи сенсоров. Состояние сенсора $\omega \in \Omega$ определяется кортежем:

$$s_\omega = \langle \Theta_\Omega, E_\Omega \rangle \tag{2.20}$$

где

$\Theta_\Omega \in \Theta$ – множество объектов МАС, воспринятых сенсором;
$E_\Omega \in E$ – множество событий, сгенерированных сенсором в текущий момент моделирования.

Корректировка модели поведения осуществляется за счет адаптации структуры дерева к текущим целям агента. Метод адаптации дерева поведения основан на добавлении и удалении узлов целей из дерева при наступлении соответствующих событий. События генерируются сенсорами агента и обрабатываются узлами реакций. Все события $e_\Omega \in E_\Omega$, наблюдаемые агентом, являются атомарными и описываются кортежем

$$e = \langle \theta_E, P_E \rangle \tag{2.21}$$

где:

$\theta_E \in \Theta$ – объект МАС, источник события;
P_E – множество именованных параметров события.

Каждое событие имеет *источник* – объект окружающей среды, который являлся причиной события. Например, если в область видимости агента попадает объект «ресурс», то становится источником события «замечен объект». Изменение значений переменных может приводить к наступлению событий, в этом случае источником события считается объект, ассоциированный с переменной. Агент сам по себе так же может являться источником события.

Основными обработчиками событий являются узлы реакций и целей в расширенном дереве поведения агента.

Каждый узел дерева поведения может обрабатывать только определенные события. Узлы реакций обрабатывают все события, которые подходят под условия фильтров, в то время как узлы целей подписываются на события, источником которых является объект, ассоциированный с целью. Подписка на событие осуществляется в момент инициализации узла. Схема обработки событий изображена на рисунке 2.6.

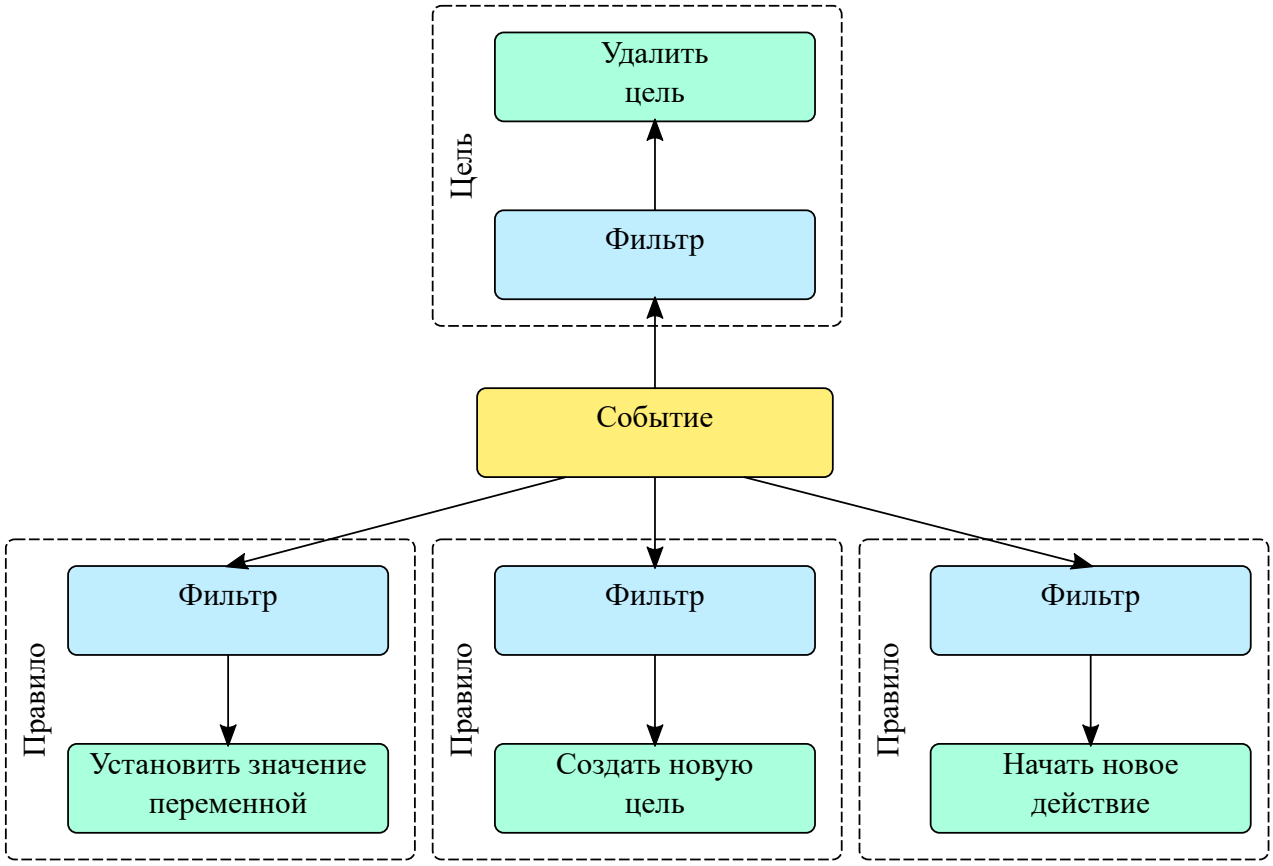

Рисунок 2.6 — Сценарии обработки события

Если узел цели принимает решение о том, является ли цель актуальной после наступления события, то узел реакций явным образом модифицирует состояние агента, для чего может выполнить одно из следующих доступных действий:

– немедленное начало выполнения действия;
– генерация цели заданного типа;
– изменение значения переменной в памяти агента;

Для полного соответствия требованиям агент должен уметь генерировать новые цели в процессе функционирования. Создание новых действий при помощи дерева решений необходимо для реализации таких реакций агента, когда

действие необходимо выполнить незамедлительно. Обычно такие действия связаны с переходом агента из одного режима функционирования в другой, например при обнаружении противника во время патрулирования агент должен немедленно оповестить соседних агентов.

Создание новых целей является основной задачей дерева решений. Предполагается, что создаваемая цель всегда ассоциирована с источником события. Одно и то же событие может приводить к созданию нескольких целей, ассоциированных с одним и тем же объектом. Конфликт между целями решается на уровне корневого узла дерева при помощи функции приоритета, косвенно учитывающей способности и ресурсы агента.

Для всех событий, источниками которых выступают объекты виртуального мира применяется метод восходящей маршрутизации (bubbling). Событие может иметь несколько объектов-проводников. *Проводником события* будем называть объект, обрабатывающий событие и инициирующий его дальнейшее распространение. Проводники образуют иерархическую систему. На самом нижнем уровне иерархии находится объект, являющийся источником события. Следующим проводником является некоторая группа объектов - например, объекты ограниченные областью. Каждый последующий проводник иерархически объединяет проводники более низкого уровня. Маршрутизация событий позволяет агенту прослушивать все события, инициированные произвольным объектом из некоторого множества. Схема маршрутизации изображена на рисунке 2.7.

2.8 Управление действиями агента

Поведение агента внешне проявляется в виде последовательно выполняемых действий, которые служат для достижения целей агента. В дереве поведения действия представлены листовыми узлами. Узлы, прикрепленные непосредственно к корневому узлу, являются реакциями на события.

Процесс выполнения агентом действий с успехом можно представить в виде простого конечного автомата из двух состояний: движения к цели и взаимодействия с объектом действия[96]. Для большей детализации декомпозируем действие агента на стадии – перемещение, подготовку, применение эффектов,

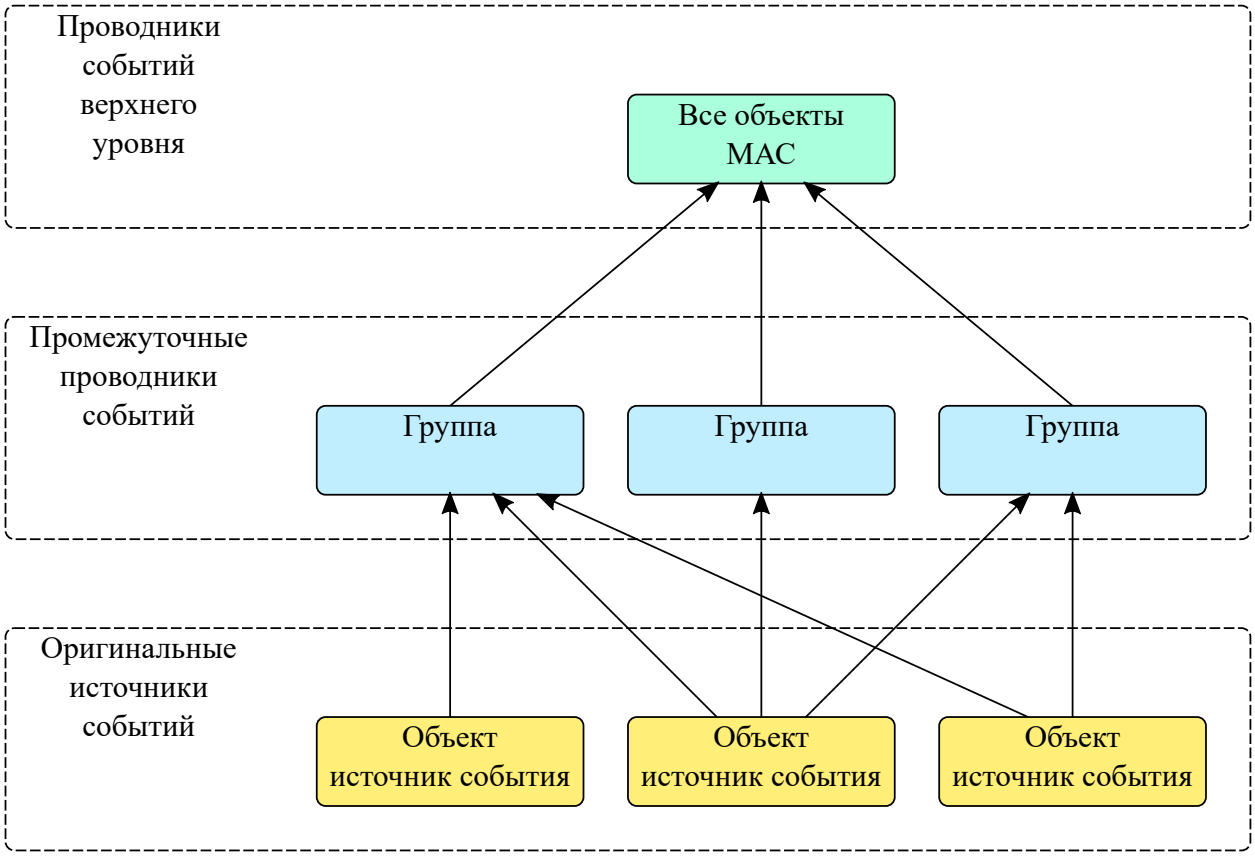

Рисунок 2.7 — Схема маршрутизации событий

завершения. В рассматриваемом классе задач можно считать, что все эффекты действий применяются мгновенно, а стадии подготовки и завершения используются для моделирования дискретной системой принятия решений непрерывных процессов, таких как управление приводами робота или анимация персонажа видео-игры. В целях упрощения можно считать продолжительность выполнения действий фиксированной во времени.

Действия выполняются приводами агента. В каждый момент времени привод может быть занят выполнением не более одного действия. Если агент имеет несколько приводов, то он соответственно может выполнять несколько действий параллельно.

Каждое действие агента может быть представлено кортежем:

$$A = \langle \theta_A, \delta_\theta, N \rangle \tag{2.22}$$

где

$\theta_A \in \Theta$ – целевой объект;
$\delta_\theta : S^2_\Theta \to S^2_\Theta$ – эффект действия;
N_A – цель навигации;

Под эффектом мы можем понимать функцию, модифицирующую свойства целевого объекта и агента-исполнителя. Выполнение действие может начаться только тогда, когда Процесс выполнения действий реализуются алгоритмом 2 в процессе обхода дерева поведения.

Алгоритм 2 — Выполнение узла действий

```
function ExecuteActionNode(action)
    switch action.state do
        case Movement:                                  ▷ Стадия перемещения
            if IsCanceled(action) then
                return Failed
            else if IsReached(action.navTarget) then
                action.timer ← now + action.prepareTime
                action.state ← Preparation
            end if
        case Preparation:                               ▷ Стадия подготовки
            if IsCanceled(action) then
                return Failed
            else if action.timer ≤ now then             ▷ Применение эффектов
                ApplyEffects(action)
                StartCooldown(action)
                action.state ← Finalization
                action.timer ← now + action.finalizationTime
            end if
        case : Finalization
            ▷ Стадия завершения
            if IsCanceled(action) ∧ action.timer ≤ now then
                return Success
            end if
    return Running
end function
```

2.9 Навигация и перемещения агента в пространстве

Планирование персонажами перемещений и избегание столкновений производится системой навигации агента и является важным элементом эффективного функционирования агента. Для взаимодействия с объектами окружающей среды агент должен находится в пределах определенной области пространства в окрестности целевого объекта. Чтобы занять необходимую позицию, агент должен совершить перемещение в пространстве виртуального мира. Для каждого способа взаимодействия может быть определено свое необходимое взаиморасположение агента и объекта. Например, чтобы открыть шкаф NPC в видео-игре должен находиться перед его дверцей, в то время как для того, чтобы подобрать предмет с поверхности достаточно подойти на некоторое расстояние с любой стороны.

Для определения желаемой позиции агента в пространстве предлагается использовать абстрактный объект *цель навигации*. Цель навигации всегда ассоциирована с объектом МАС, с которым агент собирается взаимодействовать и описывает ограничения, определяющие допустимое и оптимальное взаимные расположения агента и объекта. При построении маршрута координаты целевой точки определяются не объектом, над которым будет выполняться действие, а целью навигации, соответствующей желаемому действию, как показано на рисунке 2.8.

Цель навигации O_{nav} определяет область в пространстве виртуального мира, в которую должен прибыть агент, а также желаемые способ и скорость перемещения. Модель цели навигации может быт представлена кортежем

$$O_{nav} = \langle \theta_{target}, f_{arrival}, f_{position}, m_{movement} \rangle \tag{2.23}$$

где:

θ_{target} – целевой объект окружающей среды;
$f_{arrival} : {S_\Theta}^2 \to [0,1]$ – функция оценки близости к цели;
$f_{position} : {S_\Theta}^2 \to \mathbb{R}^2$ – функция возвращающая абсолютное значение координат цели в пространстве окружающей среды;
$m_{movement}$ – желаемый способ перемещения.

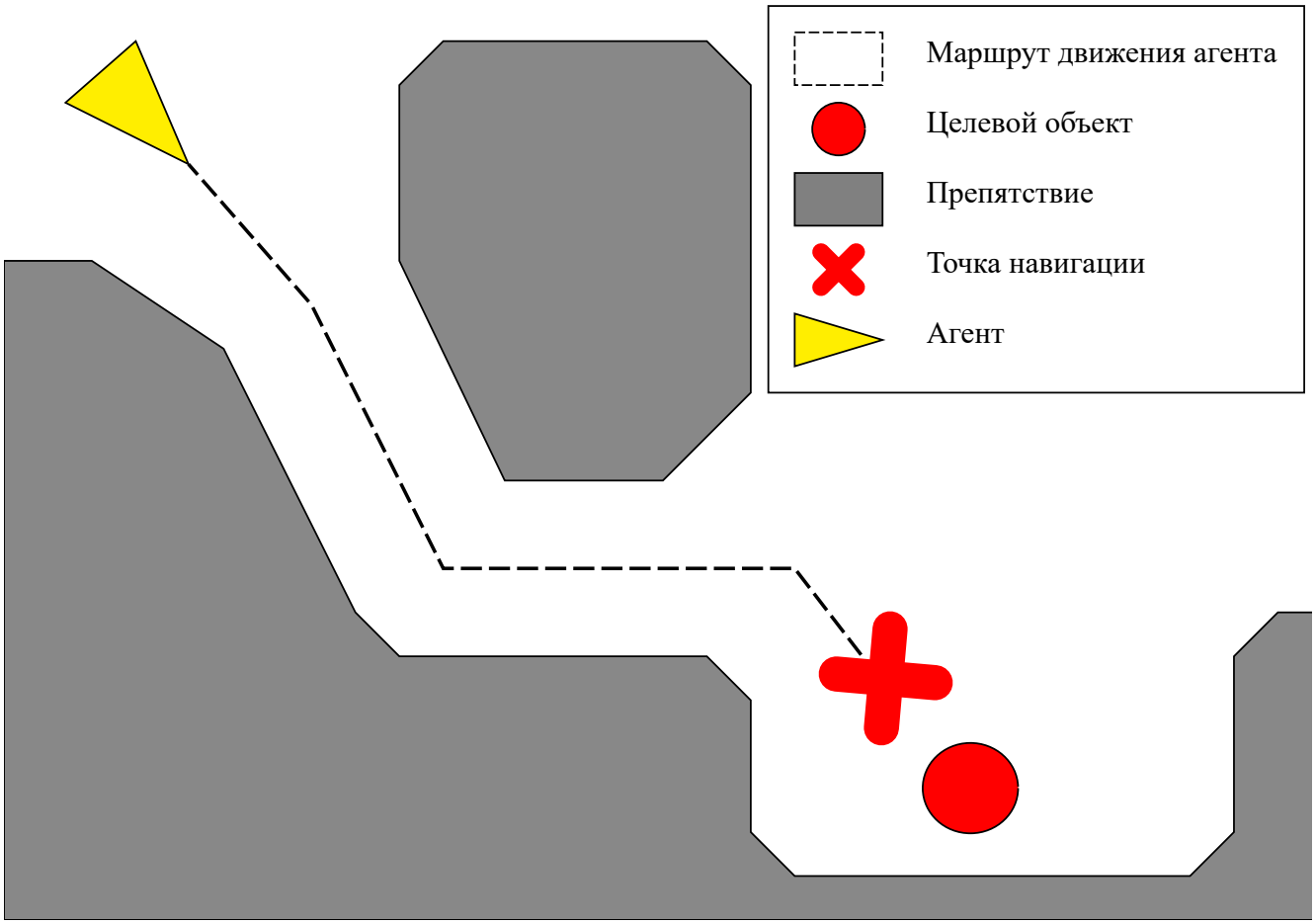

Рисунок 2.8 — Цель навигации агента

Функция оценки степени близости к цели возвращает вещественное число в диапазоне $[0; 1]$. Для любой точки за пределами области, определенной целью навигации $f_{arrival} = 0$, для точек, внутри оптимальной для взаимодействия области $f_{arrival} = 1$, для всех остальных точек значение функции равно нормированному расстоянию до границы допустимой области. Создание цели навигации осуществляется в тот момент, когда агент начинает выполнение действия. Перемещение осуществляется до тех пор, пока функция $f_{arrival}$ не вернет приемлемое значение.

Для моделирования перемещений агента используется трех-уровневая схема, включающая в себя: планирование маршрута, динамическое избегание препятствий и локомоции. Входными данными для перемещения является желаемая позиция агента, выходными – вектор собственной скорости в текущей итерации. Если агент не имеет активной цели навигации он не совершает активных перемещений и вектор собственной скорости равен нулю.

Планирование маршрута осуществляется от текущих координат агента до оптимальной целевой точки, координаты которой возвращает функция

$f_{position}$. Значение координат целевой точки рассчитываются каждую итерацию моделирования, и если расстояние между текущей и рассчитанной целевой точкой превышает указанное значение δ, новая точка становится текущей, а маршрут агента строится заново.

Избегание столкновений осуществляет корректировку маршрута агента таким образом, чтобы избежать столкновений с объектами МАС, которые не отражены в навигационном графе и игнорируются при построении маршрута. Алгоритм избегания столкновений принимает на вход текущее состояние МАС, воспринимаемое агентом, и желаемый вектор перемещения. Результатом работы алгоритма является вектор перемещения, скорректированный таким образом, чтобы движение агента по этому вектору не приводило к коллизии с другим объектом МАС. Если столкновение неизбежно, то агент должен остановиться.

На уровне *локомоций* осуществляется непосредственное перемещение агента в пространстве. Здесь же проявляется влияние окружающей среды на траекторию движения агента. В качестве примеров можно привести действие на агента гравитации или столкновения с другими объектами, которых не удалось избежать.

Процессы планирования действий и перемещений агента изолированы друг от друга с целью снижения связности системы и повышения ее сопровождаемости. Цели навигации генерируются деревом поведения и сохраняются в памяти агента. Каждой цели агента соответствует одная цель навигации, таким образом в тот момент, когда агент переключает внимание на новую цель, он автоматически меняет свою цель навигации.

2.10 Выводы

На основе формального описания поведения была разработана модульная архитектура агента. Модульность модели позволяет решить проблему мобильности и повторного использования компонентов.

Разработанная модель является гибридной. Основу модели – модуль принятия решений, составляет расширенное дерево поведения, дополненное узлами реакций и целей. Деревья поведения являются одним из наиболее производи-

тельным решением в области игрового ИИ, что обеспечивает низкое время отклика системы и позволяет добиться необходимой производительности.

Дерево поведения является динамическим и адаптирует свою структуру к изменениям в окружающей среде с помощью продукционных правил. Узлы, соответствующие неактуальным целям удаляются из дерева. Приоритет выполнения узлов дерева вычисляется с использованием многокритериальной оценки, включающей в себя полезность, ресурсозатратность и сложность достижения цели. Узлы дерева, соответствующие целям, которые обещают большую предполагаемую полезность с меньшими затратами, будут выполненны раньше. Изменения в структуре дерева поведения агента, вызванные изменениями в окружающей среде, проявляются в виде изменения модели поведения агента. Таким образом, динамическая структура решает проблему адаптивности МАС.

Предложенные эвристики позволяют оценивать приоритеты целей агента независимо друг от друга минимальным образом используя общую память. Снижение связанности компонентов модели увеличивает сопровождаемость системы и снижает стоимость ее поддержки.

Глава 3. Программная реализация мультиагентной системы

3.1 Архитектура мультиагентной системы

Программная реализация мультиагентной системы выполнена на языке C# в виде фреймворка для платформы Unity3d. Фреймворк предоставляет набор классов, реализующих работу цикла моделирования МАС, базовые элементы моделей поведения агента и инструменты визуализации.

В основе архитектуры лежит шаблон MVC, благодаря которому происходит разделение процесса моделирования и его визуализации.

Архитектура фреймворка является трехслойной и состоит из слоев данных, моделирования и представления, которые связаны между собой посредством интерфейсов. Ядро системы является сквозным, его компоненты доступны всем слоям приложения. Архитектура фреймворка представлена на рисунке 3.1.

Рисунок 3.1 — Архитектура фреймворка

Слой представления отвечает за представление результатов моделирования в режиме реального времени для пользователя. Каждый объект виртуального мира представлен на сцене объектом `GameObject` , к которому при-

креплены компоненты, необходимые для реализации различных аспектов поведения объекта МАС. Базовый компонент `MapObjectController<T>` реализует интерфейс управления объектом в цикле моделирования – инициализацию, обновление состояния, взаимодействие с другими объектами платформы Unity3d, предоставляет информацию, необходимую для работы сенсоров агентов. Данный класс параметризируется типом моделируемого объекта. Потомки `MapObjectController<T>` расширяют функциональность путем перегрузки виртуальных методов и прикрепления дополнительных компонентов – специализированных контроллеров, к объекту сцены.

Контроллеры анимации и аудио является экземплярами подкласса `MonoBehaviour` и реализует интерфейс `IMapObjectComponent<T>` . Эти компоненты прослушивают события, генерируемые объектами-моделями и предоставляют данные для компонента `Animator` в составе Unity3d.

Слой моделирования осуществляет управление объектами МАС. Слой включает в себя модель окружающей среды, представленную классом `World` , модели поведения агентов. Слой моделирования также осуществляет учет статистики. Окружающая среда разбита на локации, причем в каждой момент времени пользователь может видеть только одну локацию. Поведение объектов, расположенных на видимой пользователю локации, происходит в online-режиме с полной детализацией. Моделирование для локаций, которые пользователь не видит проходит в значительно менее детализированном offline-режиме. Объекты находящихся на offline-локациях не представлены на сцене.

Классы реализующие основные элементы окружающей среды представлены на диаграмме 3.2. Классы объектов окружающей среды реализуют интерфейс `IMapObject` . Абстрактный класс `MapObject` реализует базовые функции управелния объетками, необходимые для обеспечения доступа к объектам внутри локации.

Поведение агента на верхнем уровне представлено интерфейсом `IAgent` , который расширяет интерфейс `IMapObject` . В рамках тестового приложения, интерфейс поведения агента реализован классом `Creature` .

Характер взимодействия между агентами и интерактивными объектами виртуального мира определяется интерфейсами, которые реализую классы объектов. Персонажи могут взаимодействовать с контенерами `IItemContainer` , переключателями `IActivator` и другими объектами. Один объект может реа-

лизовывать несколько интерфейсов взаимодействия, как например двери `Door` и контейнеры `Container` .

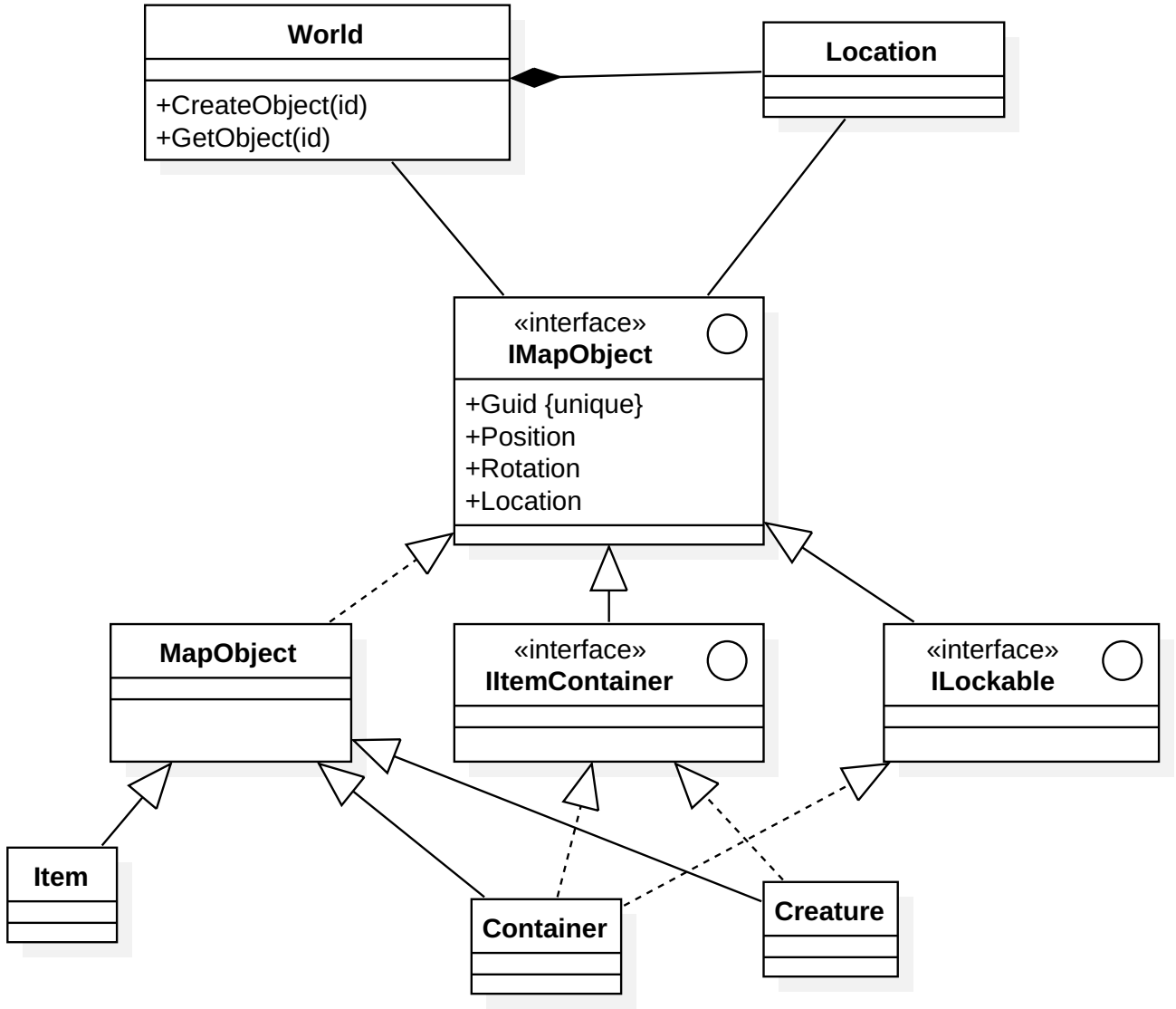

Рисунок 3.2 — Диаграмма классов. Модель виртуального мира симуляции

Слой данных предоставляет интерфейс доступа к данным, необходимым для моделирования. Динамически-загружаемые данные хранятся в виде ассетов Unity3d. К динамически-загружаемым данным относятся стандартные ассеты Unity: 3d-модели, текстуры и сцены, и дополнительные ассеты описаний объектов окружающей среда, такие как описания агентов `CreatureAsset` , ресурсов `ItemAsset` .

Статические данные опеделяют структурные свойства моделей МАС, поределяемые на этапе компиляции. К статическим данным относятся параметры агентов. Для описания параметров используются объекты-дескрипторы. Каждый объект-дескриптор имеет свой уникальный идентификатор среди объектов

своего класса. Дескрипторы определяют тип параметра и значение, используемое по умолчанию. Использование дескрипторов позволяет добиться при описании МАС гибкости, присущей динамическим языкам программирования с сохранением преимуществ статической типизации C#.

Параметры объектов МАС описываются непосредственно компонентами на сцене – поставщиками данных. Поставщики данных реализцют интерфейс `IMapObjectProvider` . Абстрактный класс `AbstractMapObjectProvider` реализует его базовую функциональность и отвечает за генерацию идентификаторов объектов.

При создании объектов МАС используется принцип ленивой инициализации, при котором создание объета происходит в момент первого обращения. Использование ленивой инициализации позволяет создавать связи между объектами МАС на уровне прямых ссылок между поставщиками данных. Это позволяет улучшить контроль целостности данных на этапе разработки.

При генерации агентов бывает необходимо, чтобы каждый агент изначально имел некоторое заранее определенное множество целей. Для решения этой задачи разработаны поставщики целей `ObjectiveProvicer` , которые также являются компонентами `MonoBehaviour` и прекрепляются к тому же объету, что и поставщик данных агента `CreatureModelProvider` . Каждый поставщик целей описывает цель определенного класса и ее параметры.

Если к одному объекту сцены `GamObject` одновременно прикреплены контроллер объекта и поставщик данных, то этот объект будет использован в качестве объекта-представления.

Ядро системы реализует системные функции и предоставляет опосредованный доступ к компонентам платформы Unity3d.

Генератор моделей отвечает за создание объектов-представлений на сцене для каждого объекта-модели для загружаемой локации. Генерация осуществляется в соответствии с алгоритмом 3. В процессе моделирования объекты МАС могут быть перемещены на другую локацию или уничтожены, однако на сцене все еще могут оставаться поставщики-данных, соответствующих этим объектам или их визуальные представления. В задачу генератора моделей входит удаление таких объектов со сцены.

Визуализация объектов, видимых пользователю должна быть максимально точной и детализированной. Для повышения детализации часто применяют

Алгоритм 3 — Алгоритм генерации объектов-представлений для локации

```
procedure LoadLocation(location)
    LoadScene(location.Scene)
    providers ← FindObjectsOfType<IMapObjectProvider>
    for provider ∈ providers do
        existed ← FindObjectWithUid(provider.uid)
        if existed = null then
            model ← CreateModel(provider)
            InitModel(model, provider)
        else if IsDestroyed(model) ∧ model.location ≠ location.uid then
            Destroy(provider)
        else
            controller ← FindController(provider)
            if controller = null then
                controller ← Instantiate(model.Prefab)
            end if
            SetModel(controller, model)
        end if
    end for
end procedure
```

технологию Root Motion, которая позволяет перемещать в пространстве объект сцены в соответствии с проигрываемой анимацией. Помимо анимации другие компоненты, такие как `Rigidbody` и `NavMeshAgent` также могут изменять координаты объекта-представления. Однако, слой моделирования ничего не знает об объектах сцены, что может привести к рассинхронизации между моделью и представлением. Для решения этой проблемы используются метод внедрения зависимостей – данные, которые могут быть изменены внутри слоя представления предоставляются модели с помощью поставщиков сервисов.

В каждом режиме моделирования (online и offline) используются соответствующие классы поставщиков сервисов. По умолчанию объект-модель использует offline-реализацию. Замена сервиса происходит в момент инициализации или уничтожения объекта-представления. В качестве примера можно привести поставщика координат объекта, диаграмма классов для которого представлена на рисунке 3.3

В режиме offline используется реализация `OfflinePositionSource` , которая является простым контейнером для координат и кватерниона вращения объекта. `OnlinePositionSource` делегирует хранение данных компоненту

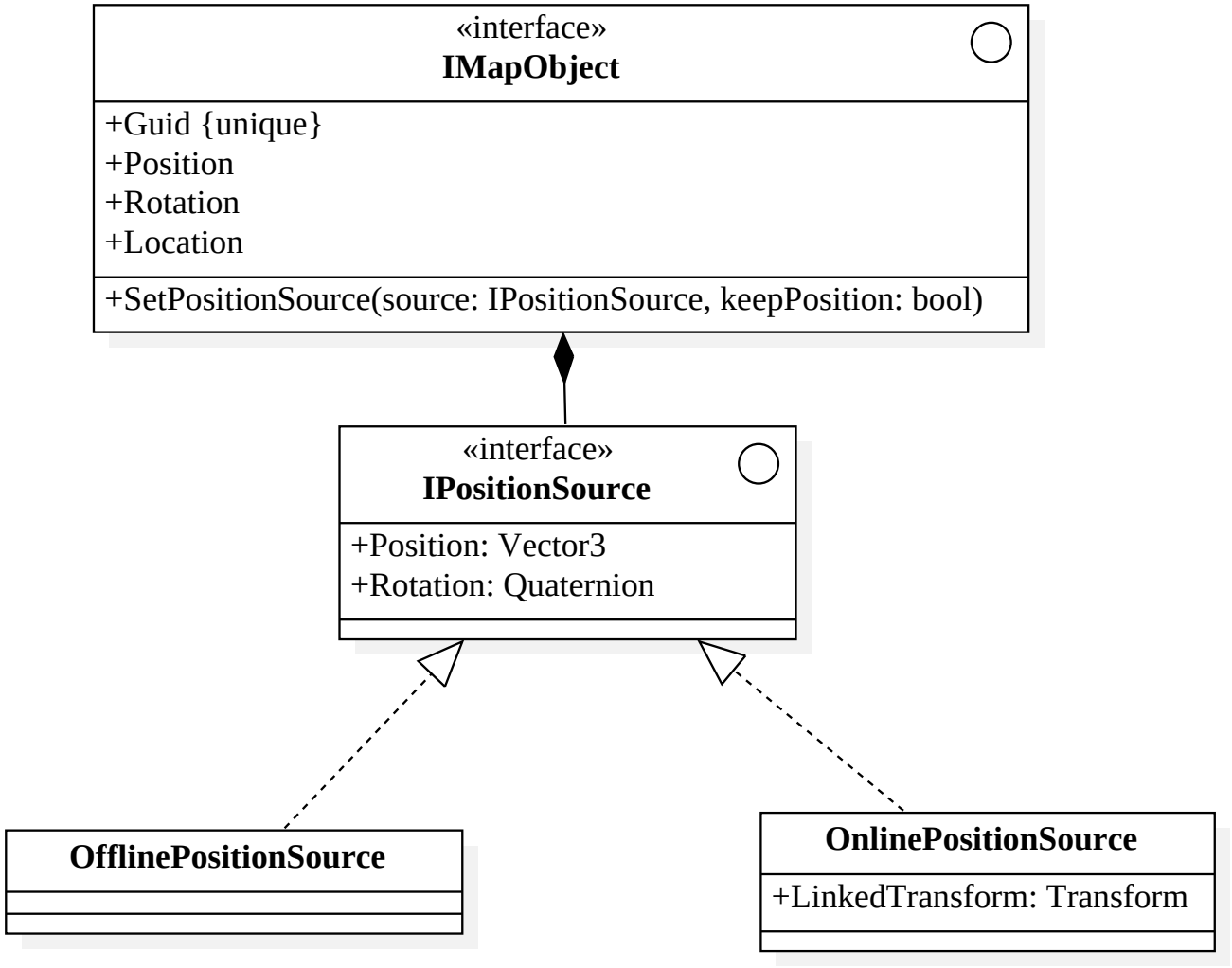

Рисунок 3.3 — Диаграмма классов. Поставщик координат

`Transform` из библиотеки Unity3d. Таким, образом любые изменения координат объекта, вызванные сторонним кодом будут адекватно восприняты на уровне модели МАС. Подобный подход позволяет интегрировать разработанный фреймворк со сторонними библиотеками без изменения внутренней структуры или интерфейса.

3.2 Сенсоры агента

Для получения информации об окружающей среде агент использует сенсоры. В разработанном фреймворке реализованы наиболее часто используемые в области разработки виде-игр сенсоры: зрительный, аккустический, тактиль-

ный и тревожный. Каждый тип сенсоров воспринимает определенные свойства объектов. В любой момент времени агент может получить доступ к списку всех объектов, которые воспринимаются сенсором и прочитать эти свойства. Схема сенсоров агента представлена на рисунке 3.4

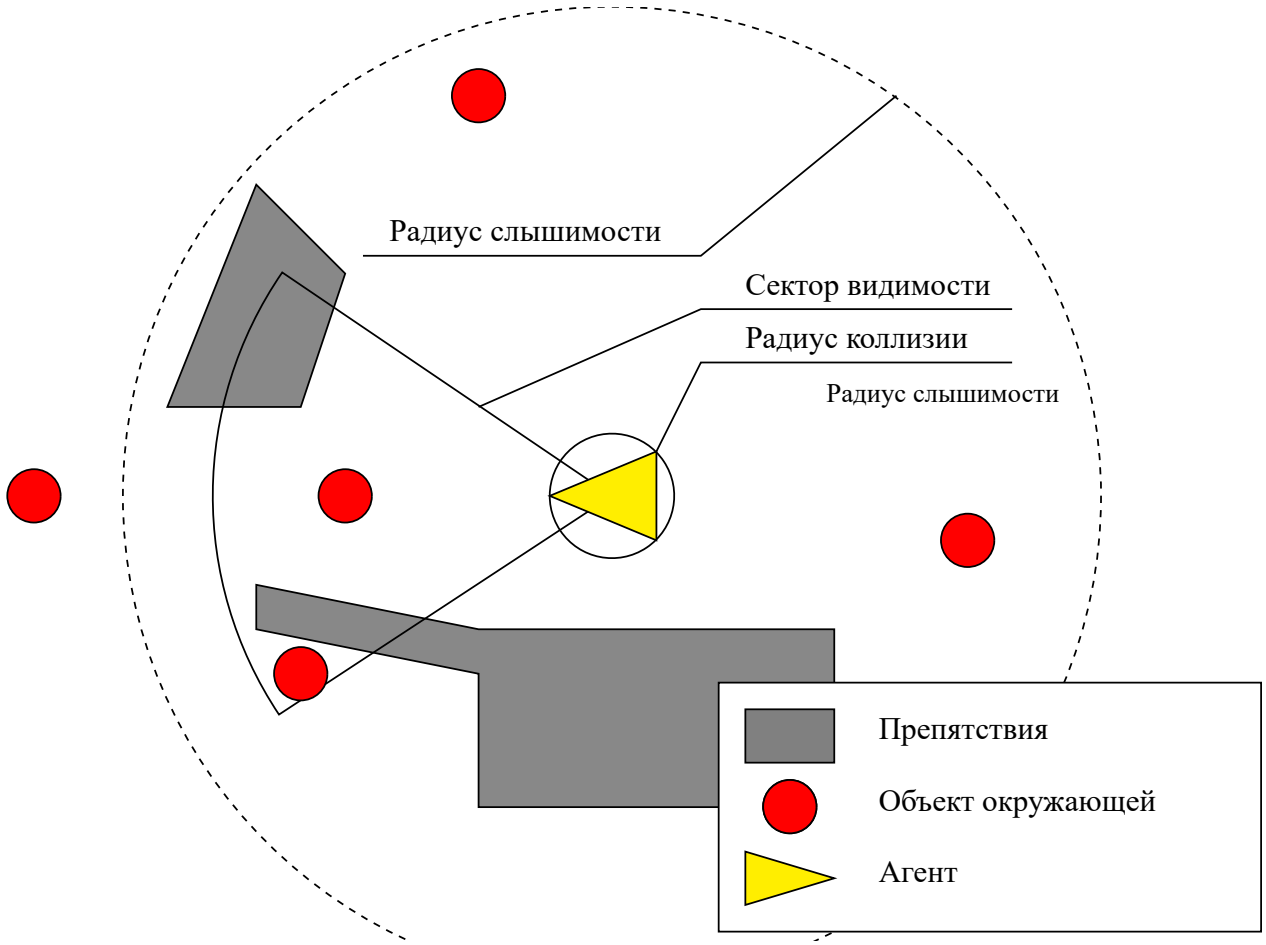

Рисунок 3.4 — Сенсоры агента

Сенсоры агента реализуют генерализованный высокоуровневый интерфейс `ISensor<T>` , предоставляющий доступ к списку воспринимаемых объектов `Objects` и события соответствующие обнаружению объекта сенсором, – `Noticed` , и его потерю, – `Lost` . Структура сенсоров приведена диаграммой классов на рисунке 3.5.

Для объектов, которые находятся на границе зоны восприятия сенсора возможно частое возникновение событий обнаружение-потеря. Во избежание излишних вычислений, связанных с их обработкой, кэшируются с использованием временных маркеров. Если объект ненадолго выходит за границу области восприятия, он все еще считается наблюдаемым. Событие о потере объекта инициируется только по прошествии порогового времени потери. Для объектов,

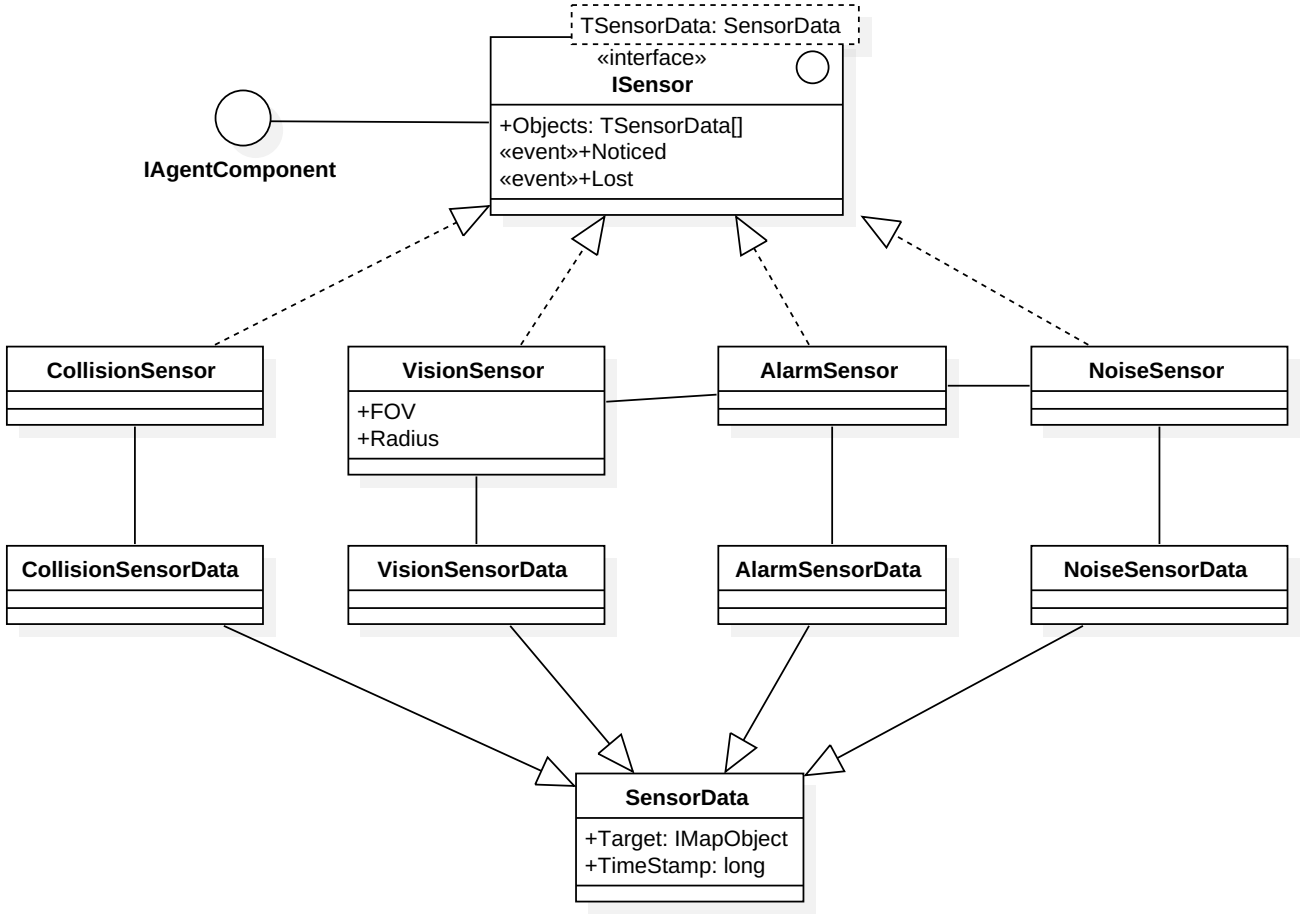

Рисунок 3.5 — Диаграмма классов. Сенсоры агента

которые физически вышли из области восприятия значения воспринимаемых параметров не обновляются.

Основным сенсором агента является виртуальный зрительный сенсор `VisionSensor` , который позволяет получить доступ к координатам объектов, попавших в область восприятия, а также идентифицировать объекты. Сенсор является виртуальным потому, что не занимается действительным распознаванием зрительных образов, а лишь осуществляет запросы к сцене Unity3d. Реализация сенсора использует инфраструктуру платформы Unity3d для получения списка всех объектов в указанном радиусе, после чего осуществляет фильтрацию по признаку попадания в конус видимости.

Сенсоры могут взаимодействовать между собой. Так тревожный сенсор `AlarmSensor` , напрямую использует зрительный и аккустический сенсоры для того, чтобы предупреждать агента о возможных опасностях и изменять его уровень тревоги. В нормальном состоянии агент предпочитает использовать мирные действия, в то время, как в настороженном – агрессивные. Уровень тревоги уменьшается, если агент не видит ни одного агрессивно настроенного агента,

и увеличивается, если таковой попадает в поле зрения. Если агент атакован, его уровень настороженности мгновенно увеличивается до максимума. Сенсор возвращает множество всех объектов, которые влияют на уровень тревоги.

Виртуальный аккустический сенсор `NoiseSensor` моделирует слух агента. Считается, что при выполнении действий объекты MAC издают шум определенной громкости и с некоторой семантикой. Если источник шума находится за пределами области видимости агента, то источником считается некоторый виртуальный объект, который создается сенсором, но не регистрируется в MAC. Аккустический сенсор позволяет реализовать реалистичное патрулирование локации агентом.

Тактильный сенсор `CollisionSensor` воспринимает физические столкновения агента с другими объектами и окружающей средой – контейнерами, препятствиями, ландшафтом. Сенсор позволяет получить доступ к информации об объектах, с которыми столкнулся агент. Для каждого объекта известны точка касания и нормаль к поверхности в этой точке.

Разработанная система сенсоров обладает свойством расширяемости. Представленное выше компоненты могут быть дополнены пользовательскими классами, для получения всей необходимой информации об окружающей среде, при реализации пользователем модели поведения агента в MAC. Кроме того, каждый сенсор поддерживает два способа доступа: выполнение запросов и подписка на события, что позволяет оптимизировать вычисления, связанные с использованием сенсоров, для различных ситуаций.

3.3 Подсистема навигации агента

При решении задачи навигации актуальной проблемой является выбор целевой точки и постоянная актуализация маршрута. За решение этой проблемы отвечает компонент «навигатор», представленный интерфейсом `INavigator`. Навигатор инкапсулирует в себе алгоритмы построения маршрута и избегания препятствий, предоставляя абстрактный интерфейс управления перемещениями.

В соответствии с предложенной моделью, для выбора координат и параметров перемещения агент использует абстрактный объект цель навигации, который как правило связан с некоторым объектом МАС. Примером цели навигации является точка напротив контейнера, из которого агент намеревается забрать ресурсы. При перемещении целевого объекта цель навигации также должна переместиться.

Решение задачи навигации осуществляется параллельно с другими задачами агента. Узлы дерева поведения могут в любой момент обратиться навигатору с целью узнать или установить текущую точку навигации. Структура подсистемы навигации представлена диаграммой классов на рисунке 3.6.

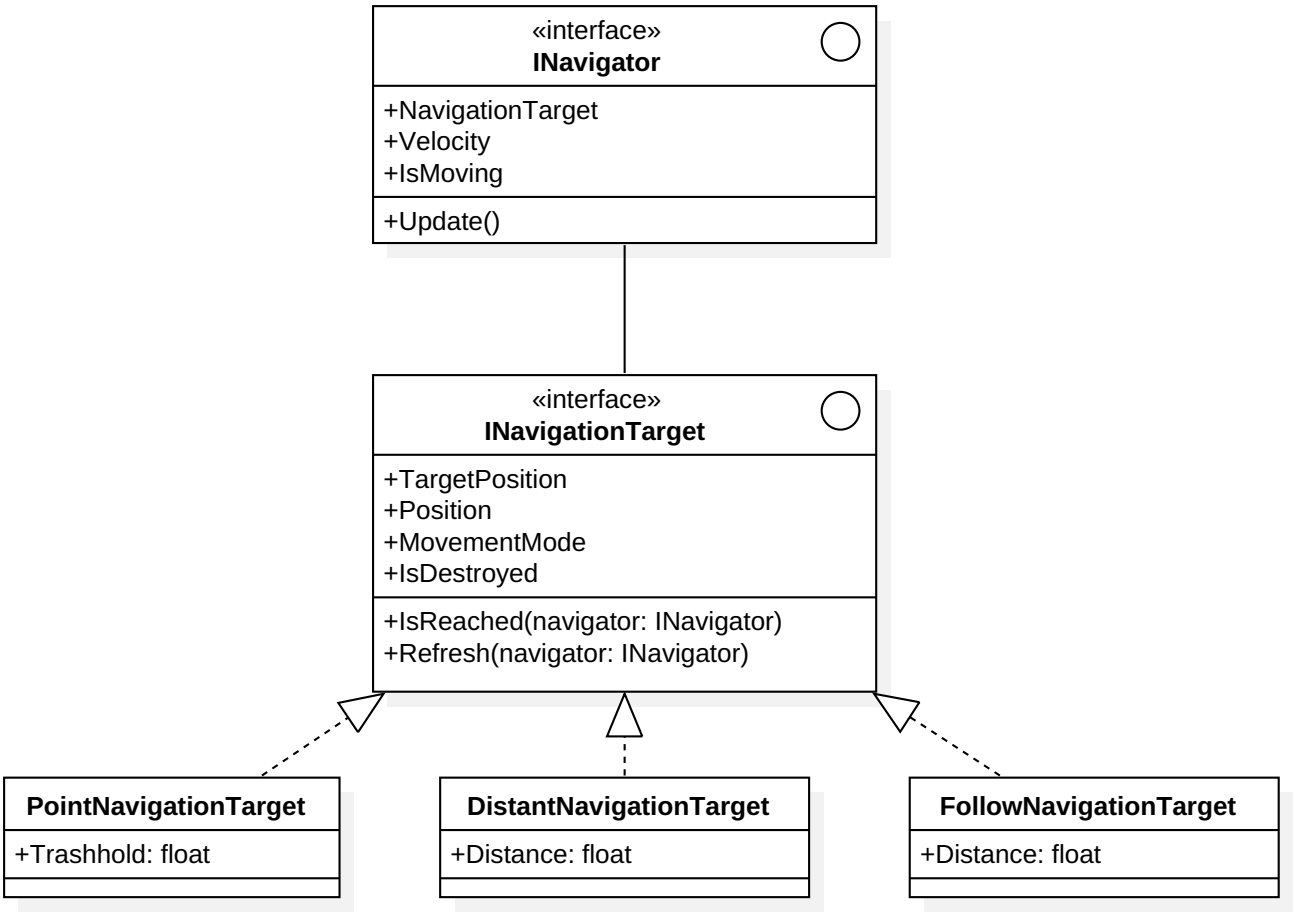

Рисунок 3.6 — Диаграмма классов. Подсистема навигации агента

Цель навигации представлена интерфейсом `INavigationTarget` . Реализующие его классы определяют параметры перемещения и функции, оценивающие степень близости агента к области, в которую нужно совершить перемещение. К параметрам перемещения относятся желаемая скорость и способ движения. Фреймворк реализует несколько типовых, наиболее распространен-

ных целей навигации: прибытие в точку, следование за целью, и взаимодействие на указанном расстоянии. Класс `DistantNavigationTarget` выбирает точку, которая находится не дальше заданного радиуса от целевого объекта. Класс `FollowNavigationTarget` позволяет агенту следовать за целивым объектом, выбирая точку перемещения в указанной окрестности позади него. Предложенный подход позволяет определить множество моделей перемещения не меняя архитектуры агента.

В зависимости от того, осуществляется визуализация перемещений для агента или нет, агент использует различные алгоритмы перемещения, реализованные соответствующими классами навигаторов. При переходе в режим Offline, используется упрощенная модель перемещений на базе алгоритма A* и навигационного графа локации состоящего из маршрутных точек. Маршрутные точки и переходы, представленные классом `Door`, образуют глобальный навигационный граф, позволяющий агентам перемещаться по всему виртуальному миру. Навигатор `OfflineNavigator` перемещает агента по маршрутным точкам внутри локации с постоянной скоростью без использования сглаживания маршрута и без учета возможных столкновений. Считается, что столкновения в режиме offline не оказывают влияния на импульс объекта.

Для перемещений в режиме online, маршрут строится с помощью алгоритма A* на навигационной сетке с высокой детализацией. Результатом работы алгоритма является ломанная линия, соединяющая целевую точку и точку с текущими координатами агента. Построенный маршрут P сглаживается алгоритмом, использованным компанией Valve в Left4Dead [43]. Каждую итерацию агент выбирает точку $p_i \in P$ с максимальным индексом i такую, что отрезок соединяющий точку, в которой находится агент и точку p_i не пересекает статических препятствий. Вектор $v_d = normalized(p_i - p_a)$ соответствует желаемому направлению движения. После применения сглаживания маршрут остается близким к оптимальному, но визуально представляется более естественным.

В процессе перемещения по такому маршруту агент может столкнутся с динамическими препятствиями – достаточно крупными объектами окружающей среды, которые необходимо обойти. Избегание столкновений осуществляется алгоритмом ORCA, основанным на скорости сближения пар объектов. Для оптимизации поиска потенциальных столкновений применяется k-d-дерево. Выходом алгоритма ORCA является скорректированный вектор скорости агента.

Финальные линейное и угловое ускорение определяются на основании свойств агента – максимальных ускорения и скорости. Локомоции агента в режиме Online реализованы при помощи компонента `CharacterController` библиотеки Unity3d. В этом случае физической моделью агента является твердое тело в форме капсулы. К капсуле применяется суммарный импульс от взаимодействия с другими физическим объектами, действия силы гравитации и ускорения агента. Контроль за перемещением капсулы делегируется физическому решателю.

Разработанная подсистема навигации позволяет гибко настраивать модель поведения агента. Алгоритм выбора точки перемещения оказывается изолированным от алгоритмов принятия решений и построении маршрута, что положительным образом сказывается на сопровождаемости и расширяемости системы.

3.4 Память агента

Память агента может быть представлена как ассоциативная коллекция, хранящая пары ключ-значение. Такая модель представления данных является стандартной для объектно-ориентированного проектирования и распространенных систем игрового ИИ. Принципиальным отличием интеллектуальных программных систем является то, что они оперируют знаниями, а не данными. Знания подразумевают наличие определенных связей между сущностями. Эти связи позволяют агенту рассуждать и делать выводы.

Построение сложных цепочек рассуждений требует значительных затрат вычислительных ресурсов, поэтому в предложенной модели предусмотренны лишь простейшие связи и зависимости. Агент может ассоциировать с каждым объектом множество фактов – свойств, и в дальнейшем использовать их в процессе принятия решений. Такие ассоциации позволяют организовать и структурировать информацию, воспринятую агентом.

Все ассоциации имеют конечное время жизни, в течении которого они актуальны. По истечении этого времени они «забываются» – навсегда стираются из памяти агента. Так например, если агент обследовал определенный фрагмент локации и забрал содержимое найденного контейнера, то контейнер будет

помечен как исследованный и на протяжении длительного времени не будет представлять интереса.

Структура памяти агента представленна диаграммой классов на рисунке 3.7. Каждая переменная в памяти описывается дескриптором `MemoryVariableDescriptor` , который определяет ее имя, тип, время жизни и значение по умолчанию. Дескрипторы являются общими и неизменяемыми объектами для всех агентов в системе. Дерево поведения агента использует дескриптор для того чтобы создать и использовать собственный, уникальный для агента, *экземпляр* переменной. Экземпляры переменных хранятся в ассоциативном контейнере с доступом по двум ключам. Первый ключ – это имя переменной, второй – ассоциированный объект MAC.

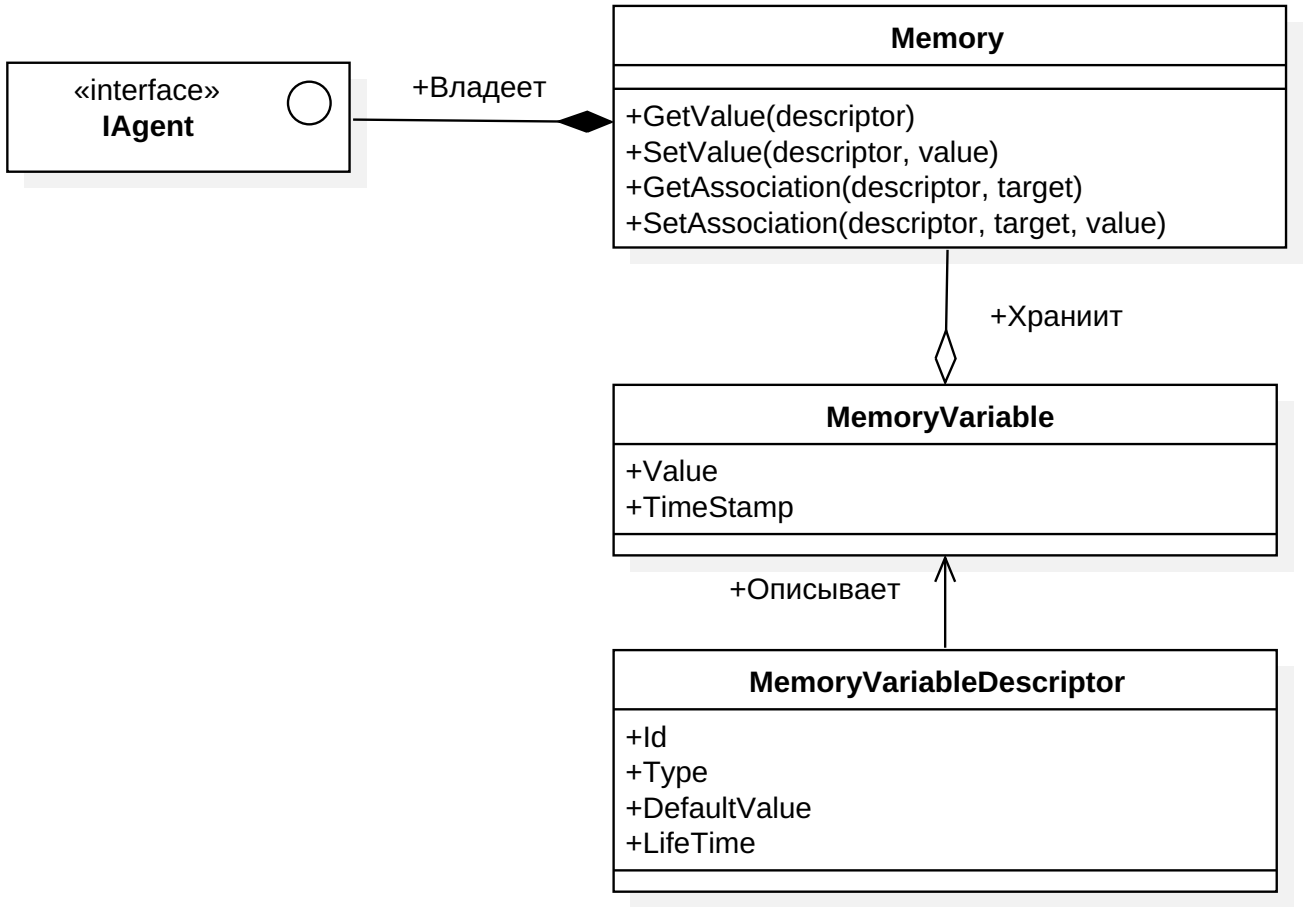

Рисунок 3.7 — Диаграмма классов. Память агента

Если обращение к переменной происходит до того, как переменная была инициализирована, или после того, как истек ее срок жизни, переменная инициализируется значением по умолчанию, которое также определено в дескрипторе. Использование дескрипторов позволяет реализовать динамическое

множество параметров для каждого объекта, при этом сохранить преимущества статической типизации и контроль над корректностью использования идентификаторов и типов данных.

3.5 Реализация расширенного дерева поведения

Разработанная модель агента основана на расширенном дереве поведения. Узлы дерева осуществляют обработку наблюдаемых событий, управляют целями агента и моделируют выполнение действий. Диаграмма классов базовых элементов дерева поведения представлена на рисунке 3.8.

Каждый узел дерева представлен интерфейсом `ITreeNode`, который позволяет получить текущее состояние узла. Логика работы узла реализуется методом *`OnExecute`*.

Связь разрозненных узлов осуществляется через композитные узлы `ICompositeNode`, каждый их которых хранит ссылки на дочерние узлы. Способ хранения и порядок обхода дочерних узлов определяется классом композитного узла. Примерами композитных узлов являются последовательности и селекторы, входящие в состав стандартного дерева поведения.

Узлы-декораторы `IDecorator`, позволяют модифицировать алгоритм выполнения декорируемого узла. К декораторам относятся циклы, условия и инверторы.

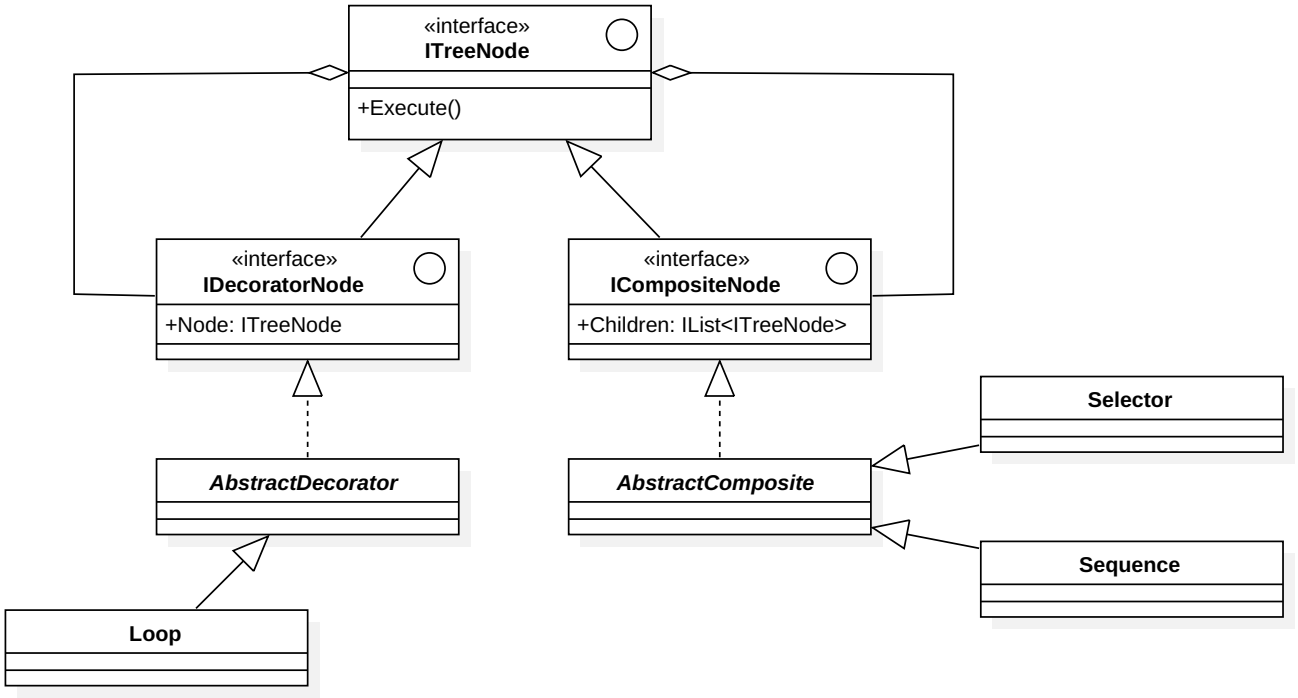

Рисунок 3.8 — Диаграмма классов. Реализация базового дерева поведения

Расширение дерева поведения реализовано в виде набора дополнительных классов. Элементы расширенного дерева поведения представленны диаграммой классов на рисунке 3.9.

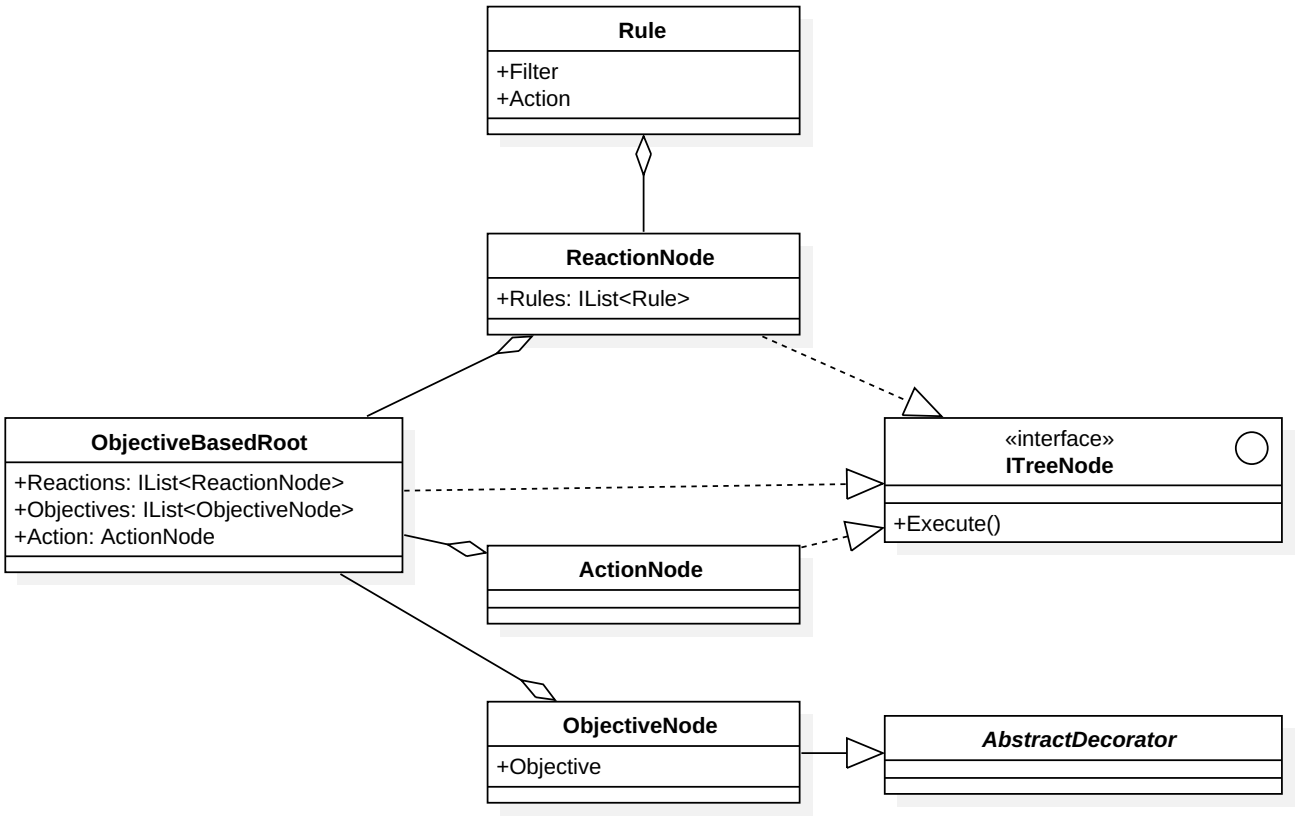

Рисунок 3.9 — Диаграмма классов. Реализация расширения дерева поведения

Корневой узел представлен классом `ObjectiveBasedRoot` . Он определяет порядок обхода остальных узлов дерева. Этот узел делегирует обработку событий узлам реакций, упорядочивает узлы целей в соответствии с приоритетом и предоставляет методы для изменения структуры дерева – добавления или удаления узлов.

Действия агента моделируются листовым узлом действий `ActionNode` . Этот узел инкапсулирует в себе способ и параметры взаимодействия агента с целевым объектом MAC.

Обработка событий осуществляется узлами реакций `ReactionNode` , каждый из которых содержит коллекцию правил обработки событий. Правила `Rule` инкапсулируют в себе фильтр события и атомарное действие, которое будет немедленно выполнено агентом. В результате выполнения правила агент может изменить значение переменной, создать новый узел цели или инициировать выполнение нового действия.

Узел цели `ObjectiveNode` инкапсулирует в себе объект цели агента. `ObjectiveNode` является декоратором, который передает управление в дочерний узел, осуществляющий непосредственно алгоритм достижения цели. Задача декоратора – предоставить интерфейс для вычисления приоритета цели, ее состояния и потенциальной достижимости в текущий момент времени.

Предложенная архитектура дерева поведения допускает создание ложных многоуровневых моделей поведения, в которых узлы `ObjectiveRootNode` являются вложенными. В этом случае каждый уровень вложенности обрабатывается независимо, и вложенные узлы будут обработаны только тогда, когда им будет передано управление.

3.6 Моделирование действий агента

Воздействия агента на окружающую среду, как правило, требуют определенного времени. Применительно к видео-играм это время обычно тратится на анимацию или обращение к удаленным вычислительным ресурсам. Любое действие агента – это некоторый процесс, результат которого проявляется не сразу. Для контроля над этим процессом разработана подсистема управления

действиями, структура которой представлена диаграммой классов на рисунке 3.10.

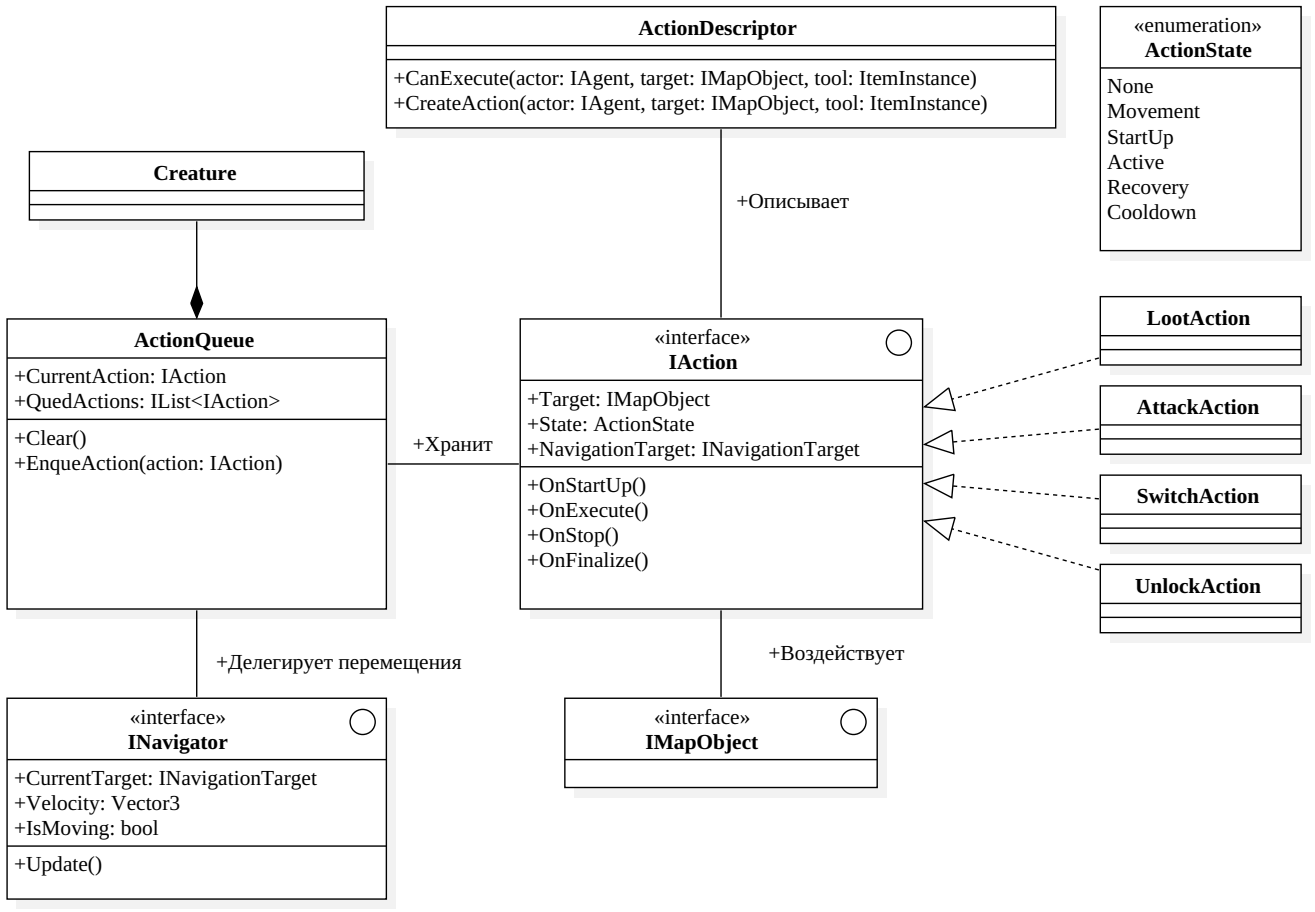

Рисунок 3.10 — Диаграмма классов. Действия агента

Все модели действий агента обязаны реализовывать интерфейс `IAction`, который обеспечивает доступ к состоянию действия. Множество возможных состояний действия определяется перечислением `ActionState`. Переходы между состояниями являются детерминированными. Если для взаимодействия с объектом агенту необходимо осуществить перемещение, выполнение действия начинается из состояния `Movement`, в котором агент устанавливает требуемую точку навигации. Если агент закончил перемещение, или если точка навигации не была назначена, действие переходит в состояние `StartUp`, для которого обычно проигрывается анимация. Влияние действия на окружающую среду осуществляется в состоянии `Active`. Это состояние может длиться всего одну итерацию моделирования, после чего действие переходит в состояние `Recovery`.

В каждый момент времени агент может выполнять не более одного, которое считается текущим на протяжении всей итерации моделирования. Если

объект намерен отменить выполнение действия, его состояние изменяется на `Recovery` , но оно все еще будет считаться текущим.

Класс `ActionQueue` обеспечивает хранение очереди действий и осуществляет передачу управления текущему действию агента. Использование механизма очереди позволяет гарантировать детерминированность начала и завершения действия, в случаях, когда дерево поведения меняет свою структуру и требует от агента переключиться на выполнение другого действия. Каждое новое действие предварительно помещается в очередь `QueuedActions` .

Агент может самостоятельно прервать текущее действие только тогда, когда оно находится в состоянии `Movement` , `StartUp` или `Active` . До тех пор, пока текущее действие находится в состоянии `Recovery` , агент не может начать выполнение нового действия. Длительность пребывания в каждом из состояний `StartUp` , `Active` , `Recovery` определяется дескриптором действия.

Дескриптор действия `ActionDescripor` является статическим объектом, определяющим общие свойства для экземпляров действий указанного класса. Дескриптор определяет принципиальную возможность выполнения агентом заданного действия, например проверяет наличие необходимых инструментов и приводов. Для того, чтобы создать новый экземпляр действия, дерево поведения также должно использовать дескриптор.

3.7 Выводы

Предложенная модель поведения агента реализована в виде фреймворка на основе Unity3d. Архитектура фреймворка содержит тремя уровня абстракции – данные, модель и представление. Слой данных предоставляет инструменты описания моделей: библиотеку базовых классов, предназначенных для определения компонентов модели. Слой моделирования реализует управление моделью МАС. На этом уровне непосредственно выполняются итерации моделирования поведения МАС. Слой представления отвечает за синхронизацию состояния модели и объектов Unity.

Разработанный фреймворк, позволяет интегрировать модель МАС и сторонние инструменты для разработки коммерческих продуктов при помощи

внедрения зависимостей. Данное решение может быть использовано для использования таких технологий как физика (Nvidia PhysiX) и Root Motion.

Разработанная система дескрипторов действий и переменных в памяти агента позволяет уменьшить связность конечного программного продукта и одновременно сохранить преимущества статической типизации языка C#. Статическая типизация, наряду с параметрическим полиморфизмом и отказом от элементов визуального программирования, делает разработанный фреймворк более устойчивым к ошибкам и более производительным чем некоторые аналогичные пакеты для Unity3d (RAIN AI, PlayMaker).

Глава 4. Тестирование и оценка качества мультиагентной системы

4.1 Описание тестового примера

Одним из целевых сценариев применения разрабатываемой модели управления агентом является реализация искусственного интеллекта в командных играх роботов. Для тестирования была разработана игра-симулятор в жанре игра на выживание, в котором участвует несколько игроков-роботов. Задача каждого робота поддерживать свое существование как можно большее время и собрать наибольшее количество ценных ресурсов. Роботы характеризуются показателями работоспособности, защиты и наносимого противнику урона. Как только показатель работоспособности станет меньшим или равным нулю, робот погибает, и агент, который им управляет выбывает из симуляции.

Для достижения своей цели роботы могут использовать предметы, которые периодически генерируются в случайных точках пространства виртуального мира. Предметы хранятся в инвентаре робота и подразделяются на экипировку, ресурсы и ценности. Экипировка увеличивает характеристики робота, такие как показатели защиты и наносимого урона. К ресурсам относятся «ремонтные наборы», которые робот может использовать для восстановления показателя работоспособности. Ценности не могут быть использованы, но их количество влияет на итоговый результат игрока. В случае гибели робота, все предметы, которые находились в его инвентаре, могут быть собраны другим роботом.

Роботы могут перемещаться в пространстве виртуального мира и взаимодействовать с другими объектами. Предусмотренны следующие виды взаимодействия: робот может подбирать предметы, атаковать других роботов, использовать источники ресурсов и предметы из инвентаря.

В соответствии с целевым сценарием опишем основные свойства и ограничения модели. В рассматриваемом примере мультиагентная система является моделью виртуального мира игры. Неподвижные объекты, такие как стены, деревья, ландшафт представлены в МАС множеством статических объектов. Персонажи, предметы и источники ресурсов – множеством интерактивных объектов.

Для каждого интерактивного объекта известны физические аспекты его модели и поведенческие реакции при различных способах взаимодействия. Взаимодействие происходит на дистанции, определяемой способом взаимодействия

и используемыми инструментами. Каждое действие робота описывается классом, реализующим интерфейс `IAction` разработанного фреймворка. В каждый момент времени агент может совершать не более одного действия.

Помимо интерактивных объектов в окружающей среде могут находиться *препятствия* – статические объекты, которые ограничивают видимость агента и мешают перемещению в пространстве. Модели препятствий обладают только физическим аспектом поведения.

Физический аспект модели робота может быть упрощенно представлен вертикально-ориентированной капсулой определённой высоты и радиуса. Робот может перемещаться в пространстве виртуального мира, он обладает массой и на него действует сила гравитации.

Информацию о других агентах и окружающей среде, агент воспринимает посредством сенсоров. В тестовом примере реализованы визуальный, тактильный и аккустический сенсоры. Визуальный сенсор определяет, какие объекты и события агент «видит». Область видимости представляет собой сектор сферы. Тактильный сенсор реагирует на столкновения с объектами окружающей среды и другими агентами. Аккустический сенсор реагирует на события, которые происходят в радиусе, превышающим радиус видимости. Сенсор воспринимает только координаты события и его семантику – шаг, падение, звук выстрела.

Поведение роботов, управляемых агентом, не должно качественно отличаться от поведения свойственному человеку. Соответственно агент должны обладать теми же возможностями и ставить такие же цели. В симуляторах жанра «игра на выживание» обычно выделяют следующие цели:

– исследование виртуального мира;
– сбор определенных предметов;
– уничтожение противника;
– помощь союзнику.

Для поддержания необходимого уровня работоспособности у контролируемого робота агент должен избегать получения урона от атак другими роботами. Если работоспособность робота снижена, агент может принять решения использовать расходуемые предметы для восстановления части работоспособности. Агент должен оценивать необходимость подбора предметов и ресурсов исходя из стоимости за выполнения действия, а так же из полезности предмета.

Проявление агрессии по отношению к другим роботам не является обязательной моделью поведения. Агент может также избегать взаимодействия с другими роботами или помогать им.

Все агенты начинают симуляцию в сходных начальных условиях – с равным уровнем здоровья и одинаковой экипировкой, но в разных точках локации виртуального мира, представленной на рисунке 4.1. Ресурсы распределены по локации равномерно и частично скрыты препятствиями. Агенты не могут наблюдать персонажей и другие объекты виртуального мира, скрытых за препятствиями. Расположение препятствий и их размеры предварительно известны всем агентам.

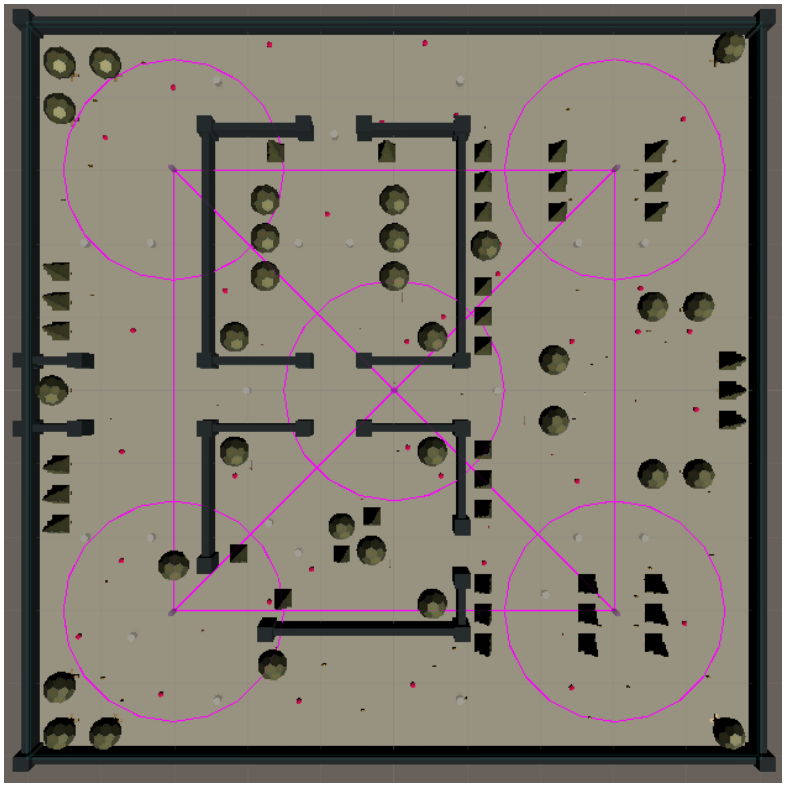

Рисунок 4.1 — Локация виртуального мира

Для определения качества разработанной модели управления на основе расширенного дерева поведения необходимо оценить результативность агентов, их способности к адаптации, сопровождаемость полученного решения и общую производительность MAC.

В ходе эксперимента управление роботами осуществлялось тремя различными способами – с помощью стандартного и расширенного деревьев поведения и непосредственно человеком.

4.2 Оценка результативности

Результативность агентов была измерена с помощью призовых очков, которыми награждались успешные действия агента. Задачей каждого робота являлись сбор ресурсов, помощь дружественным роботам и устранение противников. Таким образом задачей агента является максимизация количества очков за все время симуляции. Размер вознаграждения для каждого действия определен в таблице 2.

Таблица 2 — Вознаграждение агентов за выполнение действий

Действие	Вознаграждение
Уничтожение противника	100
Сбор возобновляемого ресурса	10
Подбор экипировки	20
Подбор «аптечки»	20
Сбор невозобновляемого ресурса	15

Эксперимент проводился в средах с различным уровнем конкуренции – нейтральной, смешанной и конкурентной. В нейтральной среде роботам запрещено атаковать друг друга, конкуренция между агентами ограничена сбором ресурсов. В смешанной среде агенты распределялись на две команды, разрешалось атаковать агентов команды-противника. В конкурентной среде разрешено атаковать любого другого агента.

Результаты эксперимента в нейтральной среде представлены на рисунке 4.2. Средние результат агентов слабо отличается от запуска к запуску. Результаты робота под управлением человека в большинстве случаев оказывается незначительно выше.

Максимальные результаты роботов представлены в виде графика на рисунке 4.3. Результат человека также выше в большинстве случаев. Процент побед ИИ равен $w_{neutral} = 66.667\%$. Средний и максимальный результат роботов под управлением расширенного дерева поведения оказывается выше, чем у роботов, под управлением стандартного дерева поведения, на 116.10% и 86.51% соответственно.

Средние результаты эксперимента в смешанной среде представлены графиком на рисунке 4.4. Разброс результатов ИИ несколько выше, чем в ней-

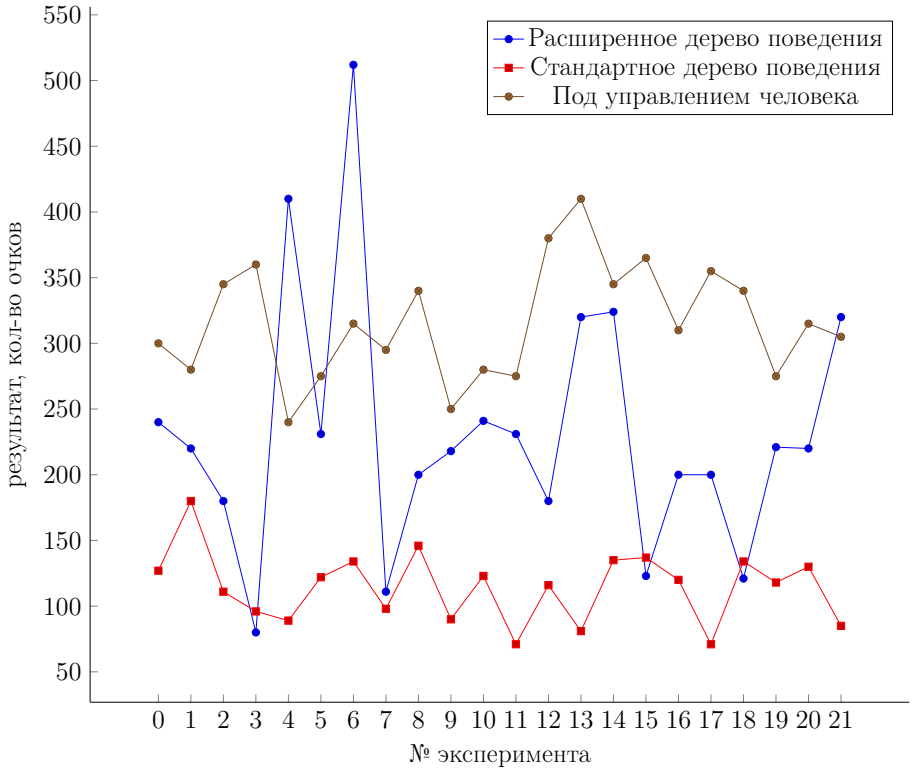

Рисунок 4.2 — Средняя результативность роботов в нейтральной среде.

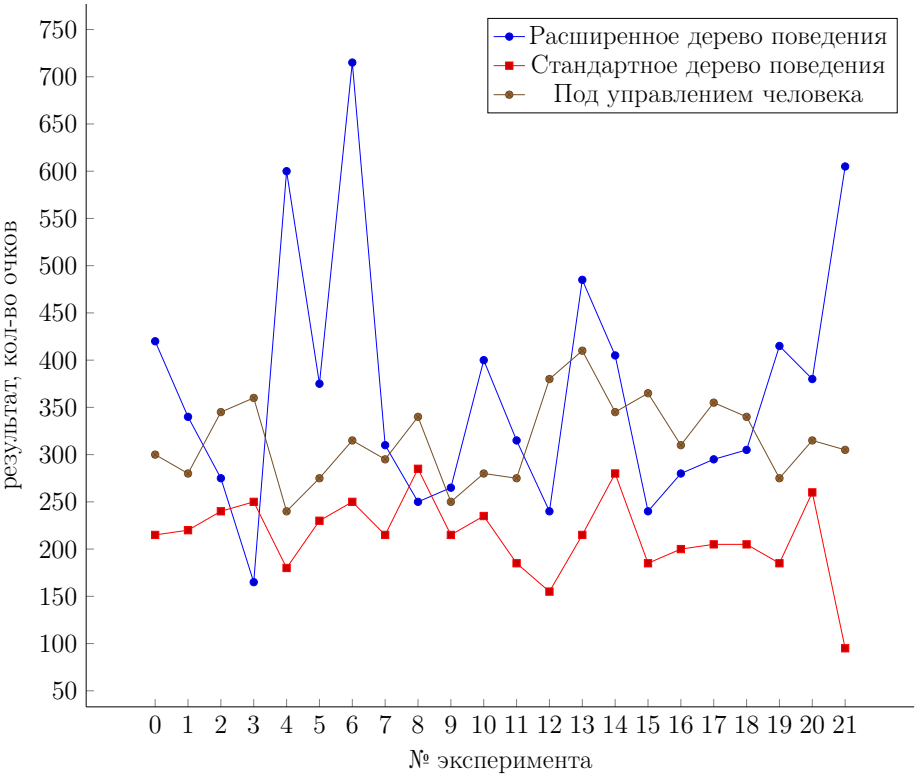

Рисунок 4.3 — Максимальная результативность роботов в нейтральной среде.

тральной среде. Увеличение разброса может быть объяснено увеличением количества различных ситуаций, в которых оказывается агент. Процент побед

среди роботов, управляемых расширенным деревом поведения, увеличивается до $w_{teams} = 47.619\%$.

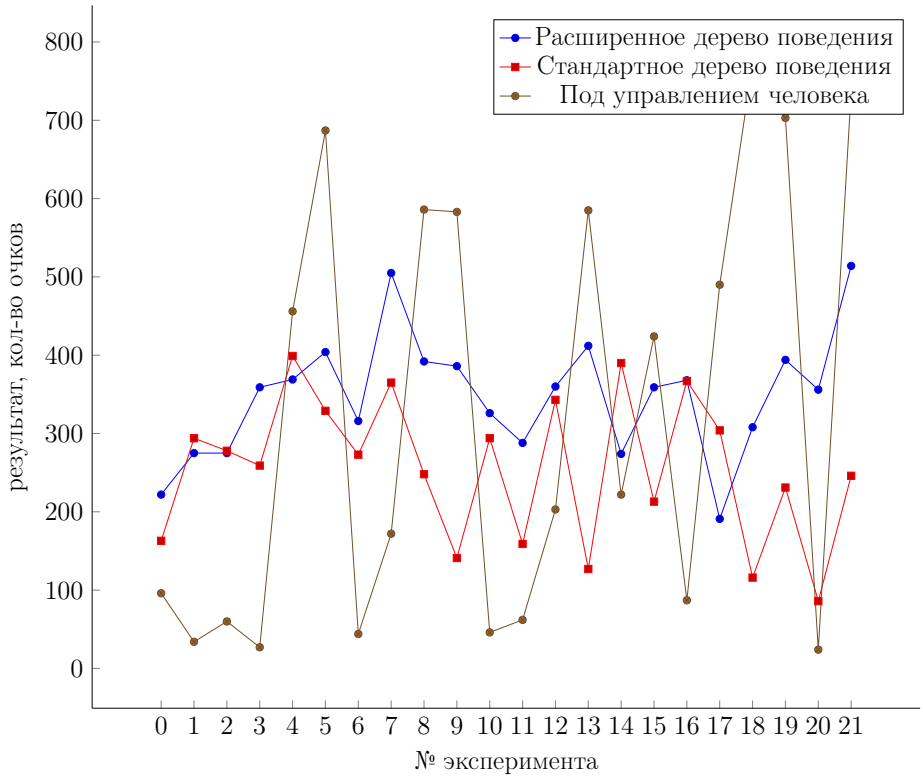

Рисунок 4.4 — Средняя результативность роботов в смешанной среде.

Максимальные результаты роботов представлены в виде графика на рисунке 4.5. Увеличение количества побед ИИ обусловлено появлением вероятности выбывания из симуляции робота под управлением человека. В смешанной среде результативность агентов на основе расширенного дерева поведения так же оказывается выше, чем у агентов на основе стандартного дерева поведения на 61.40% по среднему и на 31.96% по максимальному результату.

В конкурентой среде всем роботам разрешено атаковать друг-друга. Конкурентная среда является менее предсказуемой и требует от агентов большей адаптивности и осторожности. Средние результаты эксперимента в конкурентной среде представлены в виде графика на рисунке 4.6. На диаграмме виден более высокий разброс результатов, чем в предыдущих случаях. Количество побед ИИ также увеличивается до $w_{anarchic} = 66.667\%$, так как еще больше увеличивается вероятность выбывания робота под управлением человека.

Лучшие результаты агентов в конкурентной среде представлены в виде графика на рисунке 4.7. Результативность у агентов на основе расширенного

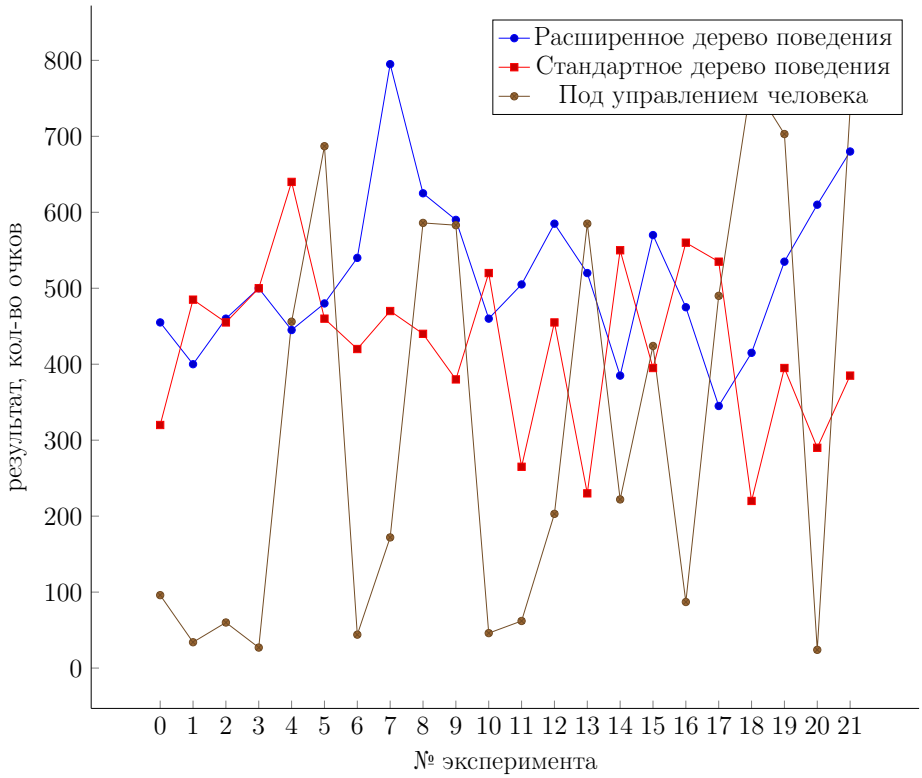

Рисунок 4.5 — Максимальная результативность роботов в смешанной среде.

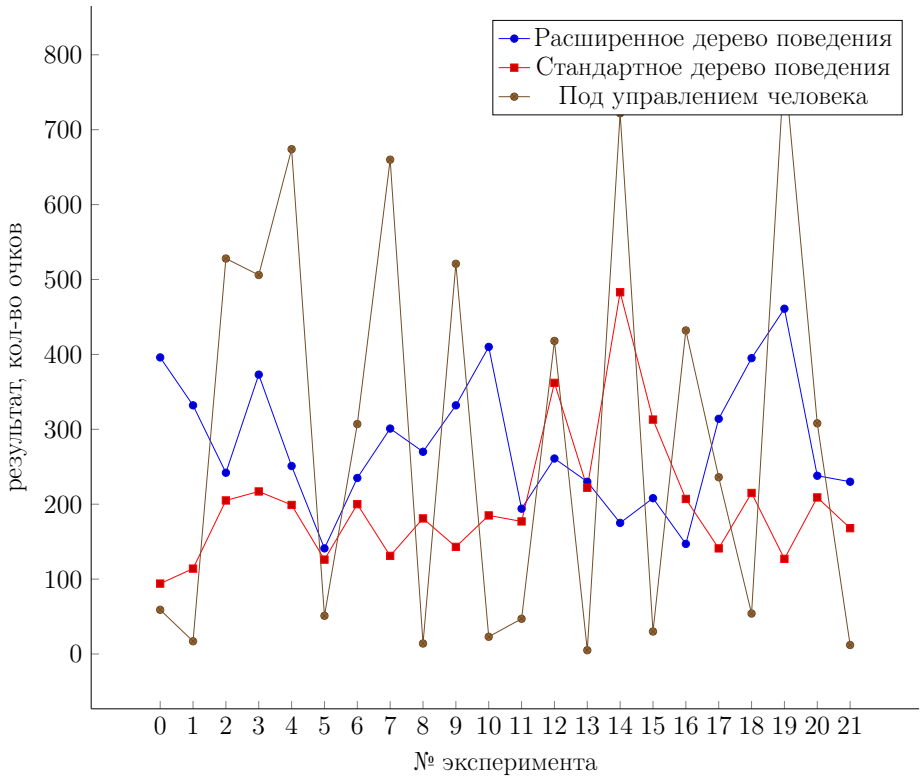

Рисунок 4.6 — Средняя результативность роботов в конкурентной среде.

дерева поведения так же выше, чем у агентов на основе стандартного дерева на 66.81% по среднему и на 36.06% по максимальному результату.

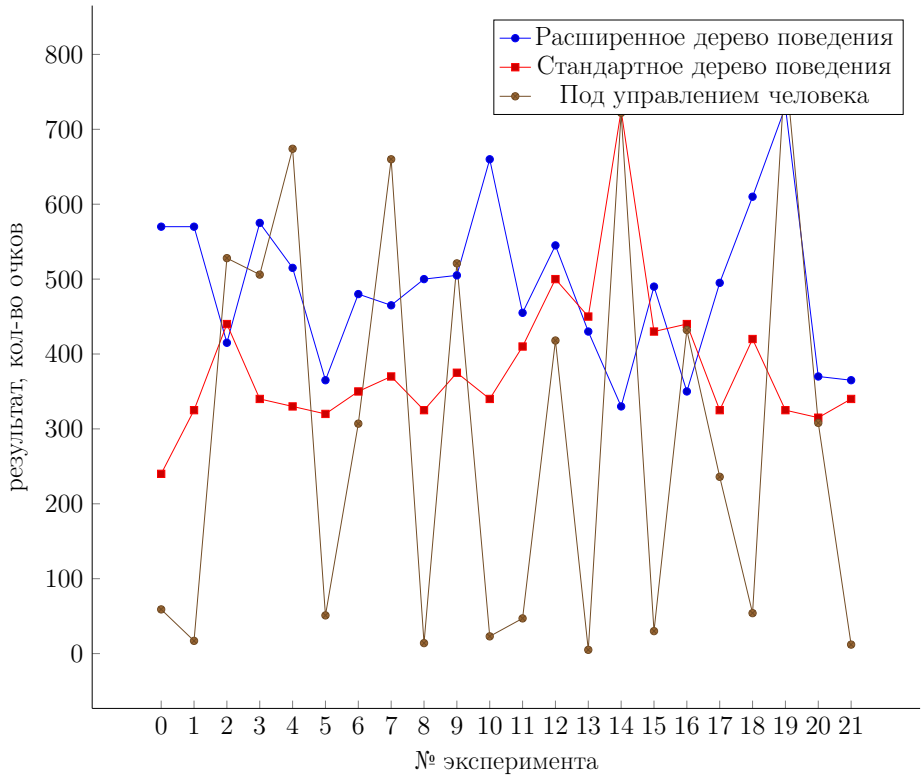

Рисунок 4.7 — Максимальная результативность роботов в конкурентной среде.
.

По результатам моделирования можно сделать вывод, что предложенная модель позволяет реализовать рациональное поведение агента. Высокий процент побед агентов на основе расширенного дерева поведения по сравнению с агентами на основе стандартного дерева свидетельствует об улучшении качества МАС и эффективности предлагаемого решения.

4.3 Оценка адаптивности

Результативность агентов зависит от характера отношений между агентами. Чем выше уровень конкуренции между агентами, тем агрессивнее они настроены по отношению друг к другу и тем большее количество различных действий могут предпринять для устранения конкуренции. Так в смешанной и конкурентной среде агентам разрешается атаковать друг друга. С ростом количества возможных действий, растет и количество принципиально различных состояний окружающей среды, которые необходимо учитывать для построения оптимальной последовательности действий.

Чем меньше изменений будет происходить в окружающей среде, тем меньшим должен быть разброс результатов. Значительное снижение результативности при увеличении уровня конкуренции свидетельствовало бы о низкой адаптивности модели поведения робота.

В результате эксперимента выявлено, что средние результаты, представленные в таблице 3, агентов под управлением расширенного дерева поведения выше, чем у агентов, управляемых стандартным деревом поведения.

Таблица 3 — Средние результаты роботов

Модель управления	Средний результат, очки		
	Нейтральная	Смешанная	Конкурентная
Расширенное дерево поведения	231.954	262.500	200.864
Под управлением человека	231.954	316.136	291.410
Стандартное дерево поведения	114.273	204.500	131

Аналогичное наблюдение можно сделать и для процента побед, представленного в таблице 4. Роботы под управлением расширенного дерева поведения остаются в состоянии противостоять роботу под управлением человека.

Таблица 4 — Процент побед роботов

Модель управления	Процент побед		
	Нейтральная	Смешанная	Конкурентная
Расширенное дерево поведения	66.667	47.619	66.667
Под управлением человека	33.333	23.810	23.810
Стандартное дерево поведения	0.000	28.571	9.523

С ростом конкуренции процент побед и результативность агентов, под управленим расширенного дерева поведения сохраняется на достаточно высоком уровне, что свидетельствует о более высокой адаптивности предложенной модели управления. Таким образом, можно сделать вывод, что предложенное решение позволяет повысить адаптивность по сравнению со стандартным деревом поведения.

4.4 Оценка производительности

Оптимальной частотой моделирования ИИ в симуляции считается 5 обновлений в секунду. Моделирование ИИ может происходить как в отдельном потоке, тогда на выполнение итерации может быть потрачено 200 миллисекунд; если моделирование выполняется в том же потоке, что и главный цикл, то на выполнение итерации должно укладываться в 15 миллисекунд.

Время выполнения итерации моделирования зависит от количества узлов в дереве поведения. Приблизительная оценка алгоритмической сложности будет иметь вид $O(N) = N \log N$. Однако, необходимо учесть, что некоторые выполняемые операции, такие как оценка приоритетов и актуальности, сами по себе могут оказаться достаточно ресурсоемкими. При относительно небольшом количестве узлов дерева, время расчета приоритетов может оказывать определяющее влияние на производительность системы.

Тестирование производительности системы производилось на машине, конфигурация которой приведена в таблице 5.

Таблица 5 — Конфигурация тестовой машины

Процессор	Intel Core i7 4700HQ 2.4ГГц
Оперативная память	16ГБ DDR3 1600МГц
Видеоадаптер	Nvidia GeForce 765M
ОС	Windows 10

Таблица 6 — Время выполнения итераций

Агрегация	Время работы (мкс)				
	Сенсоры	Навигатор	Приводы	Принятие решений	Общее
Среднее	25.3	75.4	3.4	4.9	109.6
Минимум	7.2	3.9	0.1	1.2	33.3
Максимум	12178.3	904.9	10733.2	10530.9	12372.4

Как следует из таблицы 6, время моделирования остается в границах приемлемого диапазона.

На этапе разработки модели проводилось тестирование производительности для предложенного решения и целе-ориентированных агентов. Целе-ориентированные модели поведения так же обладают высокой адаптивностью, но при этом имеют более высокую алгоритмическую сложность. Для сравнения

Таблица 7 — Распределение времени выполнения итерации

Операция	Процент
Сенсоры	23.1
Поиск маршрута	68.9
Моделирование действий	3.1
Обновление приоритетов	4.8
Другие операции	0.1

были реализованы два планирующих алгоритма на базе A*. Для оценки производительности производились замеры времени принятия решения в нескольких типовых ситуациях.

Первый алгоритм является «наивной» реализацией, совмещающей планирование перемещений и действий агента. Предполагается, что данный алгоритм будет предлагать более эффективный план действий за счет большей детализации информации.

Второй алгоритм осуществляет поиск плана выполнения одной из ближайших доступных задач. Выбор задачи для построения маршрута осуществляется с помощью «жадной» эвристики. Перемещения и действия планируются раздельно, что позволяет снизить общее время выполнения.

Тестирование проводилось в типовых ситуациях, представленных в таблице 8, с заранее заданными целями агентов. В ходе замеров времени принятия решения считалось, что моделирование носит эпизодический пошаговый характер и каждую итерацию агенты могут выполнить только одно действие.

Результаты замеров представлены в виде столбчатой диаграммы на рисунке 4.8. На диаграмме видно, что производительность предлагаемого решения значительно выше, чем у целе-ориентированных алгоритмов при решении схожих задач. Следует отметить, что в случае D детализированный алгоритм планирования не получил решения в течении 30с и был остановлен.

4.5 Оценка сопровождаемости системы

Одним из критериев качества программного продукта является качество кода. Архитектура системы должна «подталкивать» разработчика к тому, чтобы полученный продукт был как можно более качественным. К свойствам ис-

Таблица 8 — Тестовые ситуации для оценки производительности

№	Ситуация	Описание
A	Сбор предметов (полный)	Задача агента собрать все предметы, расположенные на карте.
B	Сбор предметов (приблизительный)	Задача агента – собрать все предметы, расположенные на карте. Детальное планирование перемещений исключается.
C	Уничтожение единичного противника	Запас боеприпасов – неограниченный. Считается, что противник неподвижен и не выполняет ответных действий.
D	Уничтожение единичного противника	Запас боеприпасов ограничен. Считается, что противник неподвижен и не выполняет ответных действий.
E	Уничтожение 3-х противников	Запас боеприпасов неограничен. Считается, что противники не выполняют ответных действий.
F	Уничтожение 3-х противников	Запас боеприпасов ограничен. Считается, что противники не выполняют ответных действий.

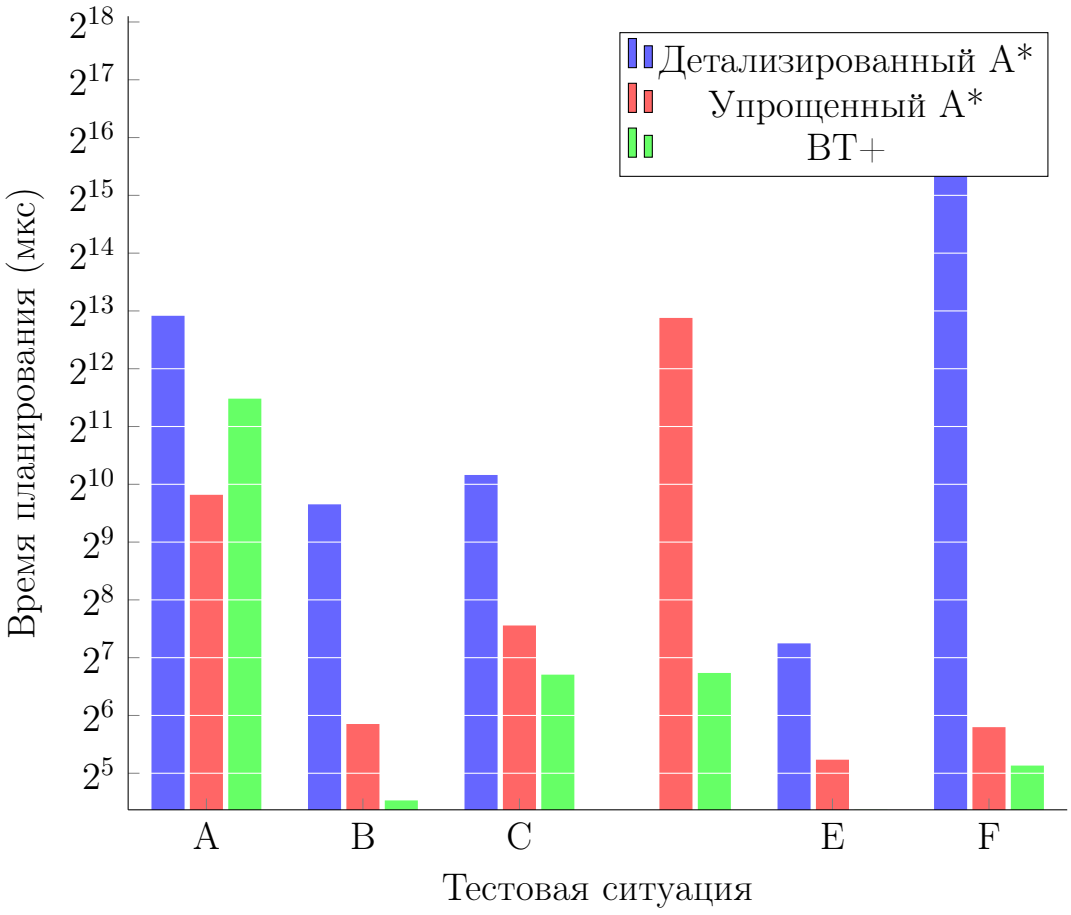

Рисунок 4.8 — Сравнение производительности

ходного кода, напрямую влияющим на качество программного продукта относят сопровождаемость и масштабируемость.

Сопровождаемость является комплексной характеристикой, оценивающей объем исходного кода, его цикломатическую сложность, количество уровней вложенности, глубину наследования и связанность компонентов ПО. Объем исходного кода в модели поведения агента снижается за счет использования типовых модулей, реализующих память агента, функции сенсоров и приводов, алгоритмов принятия решений. Уменьшение связности программы агента осуществляется за счет использования абстрактных интерфейсов и компонентоориентированного подхода к построению модели.

Для оценки сопровождаемости кода использовался разработанный компанией Microsoft индекс [97], основанный на индексе Paul Oman и Jack Hagemeister. В отличии от оригинального индекса, индекс, разработанный компанией Microsoft имеет область значений $[0; 100]$, что делает его более интуитивным и удобным в использовании.

Индекс вычислялся для классов, входящих в разработанный фреймворк и для классов, реализующих непосредственно модель поведения агента. Значение индекса в обоих случаях находится в диапазоне $[60; 90]$, что свидетельствует о низкой связанности, хорошей расширяемости и сопровождаемости предложенного технического решения.

4.6 Выводы

В ходе тестирования было выявлено, что предложенная модель управления агентом на основе расширенного дерева поведения является более эффективным решением, по сравнению с аналогичным стандартным деревом поведения. В тестовом окружении разработанные модели продемонстрировали высокую результативность и адаптивность. Результаты, продемонстрированные роботами, под управлением ИИ оказались близки к результатам робота под управлением человека, что свидетельствует о качестве принимаемых решений и рациональности предлагаемой модели.

В таблице 9 приведены результаты сравнения разработанной модели с референсными конечно-автоматными моделями (стандартным деревом поведения).

Таблица 9 — Итоговая сравнительная таблица моделей поведения агентов

Критерий	Стандартное дерево поведения	Расширенное дерево поведения
Сопровождамость	Средняя	Высокая
Адаптивность	Низкая	Средняя
Производительнсоть	Высокая	Высокая

Адаптивность модели поведения агента была повышена за счет введения узлов-реакций, модифицирующих структуру дерева поведения, и динамической функции вычисления приоритета узлов.

Предложенная модель позволила добиться увеличения сопровождаемости модели поведения агента за счет уменьшения глубины вложенности деревьев и общего количества узлов, замены глобальных переменных локальными. Низкоуровневое управление действиями и перемещениями также вынесено за пределы дерева. Предложенная эвристическая функция оценки приоритета использует управляющие коэффициенты с прозрачной семантикой, что также благоприятно сказывается на сопровождаемости модели.

Разработанная модель обладает схожей высокой производительностью, свойственной всему классу конечно-автоматных моделей. Таким образом, предложенное решение позволяет повысить общее качество конечно-автоматных МАС, за счет увеличения сопровождаемости и адаптивности моделей поведения агентов.

Заключение

Главным результатом диссертационной работы является повышение качества разрабатываемых мультиагентных систем, построенных на основе расширенного дерева поведения. Повышение качества МАС, разрабатываемых с использованием предлагаемых моделей, достигается за счет снижения цикломатической сложности, уменьшения общего количества элементов системы, увеличения сопровождаемости и адаптивности системы, по сравнению с МАС, построенных с использованием классических конечно-автоматных моделей. Агенты на основе расширенного дерева поведения в ходе тестирования продемонстрировали в среднем на 60% лучшие результаты, чем агенты на основе стандартного дерева поведения.

Предложенная архитектура программной реализации позволила снизить количество зависимостей между компонентами. Разработанные интерфейсы позволяют провести интеграцию сторонних компонентов, что повышает прикладную ценность системы.

В результате диссертационного исследования:

1. выявлены проблемы процесса разработки мультиагентных систем реального времени;
2. проведен анализ существующих моделей поведения агентов на соответствие критериям качества МАС;
3. разработана модель расширенного дерева поведения, позволяющая повысить адаптивность и сопровождаемость МАС по сравнению со стандартными деревьями поведения;
4. предлагаемые модели и алгоритмы реализованы в виде фреймворка, позволяющего сократить затраты на разработку систем агентного моделирования и игровых приложений;
5. апробирована эффективность предлагаемых моделей, методов и программных средств на примере конкурентной среды.

Таким образом, в ходе выполнения диссертационной работы были решены все поставленные задачи и достигнута цель исследования.

Перспективы дальнейшей разработки темы исследования:

1. исследование и разработка алгоритмов машинного обучения для адаптации функций оценки приоритета целей агента в процессе моделирования;
2. разработка моделей взаимодействия агентов для более эффективного выполнения задач;
3. развитие фреймворка и расширение набора программных компонентов реализации моделей поведения агентов.

Список литературы

1. Wooldridge M. An introduction to multiagent systems. — JOHN WILEY & SONS, LTD, 2002. — 484 с.
2. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. — 3rd. — Upper Saddle River, NJ, USA : Prentice Hall Press, 2009. — 1152 с. — ISBN 0136042597, 9780136042594.
3. Schelling T.C. Dynamic models of segregation // The Journal of Mathematical Sociology. — 1971. — Т. 1, № 2. — С. 143—186.
4. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. — Москва : Эдиториал УРСС, 2002. — 352 с.
5. Swarm Symulation System. — Режим доступа: www.swarm.org.
6. NetLogo. — Режим доступа: http://ccl.northwestern.edu/netlogo/.
7. Добрынин Д.А. Интеллектуальные роботы. Состояние и перспективы // КИИ-2010. — 2010. — Режим доступа: http://www.raai.org/resurs/papers/kii-2010/doklad/dobrynin.pdf.
8. Cipi E., Cico B. Simulation of an Agent Based System Behavior in a Dynamic and Unpredicted Environment // World of Computer Science and Information Technology Journal (WCSIT). — 2011. — Т. 1, № 4. — С. 172—176. — ISSN 2221-0741.
9. Вайнцвайг М.Н., Полякова М.П. О моделировании мышления // Научная сессия МИФИ-2003. Нейроинформатика-2002. 5 Всероссийская научно-техническая конференция. Проблемы интеллектуального управления-общесистемные, эволюционные и нейросетевые аспекты. Материалы дискуссии. — 2003. — С. 77—85.
10. Новиков Д.А. Иерархические модели военных действий // Управление большими системами: сборник трудов. — ИПУ РАН, 2012. — С. 25—62.
11. Millington I., Funge J. Artificial Intelligence for Games. — Morgan Kaufmann publishers, 2006. — 895 с.
12. Intelligent agents in computer games / M. Van Lent [и др.] // Proceedings of the National Conference on Artificial Intelligence. — 1999. — С. 929—930.

13. Adaptive game AI with dynamic scripting / P. Spronck [и др.] // Machine Learning. — 2006. — Т. 63, № 3. — С. 217—248. — ISSN 1573-0565.
14. Dignum F. Agents for Games and Simulations // Autonomous Agents and Multi-Agent Systems. — Hingham, MA, USA, 2012. — Март. — Т. 24, № 2. — С. 217—220. — ISSN 1387-2532.
15. Isla D., Blumberg B. New challenges for character-based AI for games // Artificial Intelligence and Interactive Entertainment II. — AAAI Press, 2002. — С. 41—45.
16. Research directions for AI in computer games / C. Fairclough [и др.] // Proceedings of the Twelfth Irish conference on artificial intelligence and cognitive science / под ред. D. O'Donoghue. — NUIM Department of Computer Science, 2001. — С. 333—344. — ISBN 0901519480.
17. Lerman K., Galstyan A. Agent Memory and Adaptation in Multi-agent Systems // Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. — Melbourne, Australia : ACM, 2003. — С. 797—803. — (AAMAS '03). — ISBN 1-58113-683-8. — Режим доступа: http://doi.acm.org/10.1145/860575.860703.
18. Xiao L., Greer D. Modeling, Auto-generation and Adaptation of Multi-agent Systems // Proceedings of the Tenth CAiSE/IFIP8.1 International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05). — Porto, Portugal, 06.2005. — С. 605—616.
19. Beer R.D., Gallagher J.C. Evolving dynamical neural networks for adaptive behavior // Adaptive behavior. — 1992. — Т. 1, № 1. — С. 91—122.
20. Gutiérrez J.A.G., Cotta C., Fernández Leiva A.J. Design of Emergent and Adaptive Virtual Players in a War RTS Game // Proceedings of the 4th International Conference on Interplay Between Natural and Artificial Computation - Volume Part I. — Canary Islands, Spain : Springer-Verlag, 2011. — С. 372—382. — (IWINAC'11). — ISBN 978-3-642-21343-4.
21. Pitt J., Mamdani A. Communication Protocols in Multi-agent Systems: A Development Method and Reference Architecture // Issues in Agent Communication / под ред. F. Dignum, M. Greaves. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. — С. 160—177. — ISBN 978-3-540-40028-8. — Режим доступа: http://dx.doi.org/10.1007/10722777_11.

22. Communicating Agents in Open Multi Agent Systems / T.R. Payne [и др.] // Proceedings of 1st GSFC/JPL Workshop on Radical Agent Concepts (WRAC). — 2002. — С. 365—371.
23. Open Multi-Agent Systems: Agent Communication and Integration / R.M. van Eijk, F.S. de Boer, J.-J.C. van der Hoek Wiebe and Meyer // Intelligent Agents VI. Agent Theories, Architectures, and Languages: 6th International Workshop, ATAL'99 / под ред. N.R. Jennings, Y. Lespérance. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. — С. 218—232. — ISBN 978-3-540-46467-9.
24. ISO/IEC ISO/IEC 25000:2014 – Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. — 2-е изд. — ISO /IEC, 2014.
25. ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению. — 1994.
26. Marir T., Mokhati F., Bouchlaghem-Seridi H. Do we need specific quality models for multi-agent systems? Toward using the ISO/IEC 25010 quality model for MAS // Software Engineering and Applications (ICSOFT-EA), 2014 9th International Conference on. — IEEE, 2014. — С. 363—368.
27. Gehrke J.D., Schuldt A., Werner S. Quality criteria for multiagent-based simulations with conservative synchronisation // 13th ASIM Dedicated Conference on Simulation in Production and Logistics / под ред. M. Rabe. — Fraunhofer IRB Verlag, 2008. — С. 545—554.
28. Measuring the Social Ability of Software Agents / F. Alonso [и др.] // Software Engineering Research, Management and Applications, 2008. SERA'08. Sixth International Conference on. — IEEE, 2008. — С. 3—10.
29. Zuse H. Software complexity. — Hawthorne, NJ, USA : Walter de Gruyter & Co., 1991. — ISBN 0-89925-640-6.
30. Stein C., Cox G., Etzkorn L. Exploring the Relationship between Cohesion and Complexity // Journal of Computer Science. — 2005. — Т. 1, вып. 2. — С. 137—144. — ISSN 1549-3636.
31. Sarkar A., Debnath N. Measuring complexity of Multi-Agent System architecture // IEEE 10th International Conference on Industrial Informatics. — 2012. — С. 998—1003.

32. Dechter R., Pearl J. Generalized best-first search strategies and the optimality of A* // Journal of ACM. — 1985. — Т. 39, № 3. — 505-536.
33. Botea A., Müller M., Schaeffer J. Near optimal hierarchical path-finding // Journal of Game Development. — 2004. — Т. 1. — С. 7—28.
34. Harabor D., Botea A. Hierarchical path planning for multi-size agents in heterogeneous environments. // 2008 IEEE Symposium On Computational Intelligence and Games / под ред. P. Hingston, L. Barone. — IEEE, 2008. — С. 258—265. — ISBN 978-1-4244-2973-8. — Режим доступа: http://dblp.uni-trier.de/db/conf/cig/cig2008.html#HaraborB08.
35. Harabor D., Grastien A. Online Graph Pruning for Pathfinding on Grid Maps // National Conference on Artificial Intelligence (AAAI). — 2011.
36. Harabor D., Grastien A. Improving Jump Point Search. Pathfinding Algorithm // Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS). — 2014.
37. Reynolds C.W. Flocks, herds and schools:A Distributed Behavioral Model // Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. — New York, NY, USA : ACM, 1987. — С. 25—34. — (SIGGRAPH '87). — ISBN 0-89791-227-6. — Режим доступа: http://doi.acm.org/10.1145/37401.37406.
38. Reynolds C.W. Steering behavior for autonomous characters // Game Developers Conference. — 1999. — С. 763—782.
39. Reynolds C.W. OpenSteer. Steering Behaviors for Autonomous Characterss. — 2002. — Режим доступа: http://opensteer.sourceforge.net/.
40. Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots. / J. Alonso-Mora [и др.] // DARS. Т. 83 / под ред. A. Martinoli [и др.]. — Springer, 2010. — С. 203—216. — (Springer Tracts in Advanced Robotics). — ISBN 978-3-642-32722-3. — Режим доступа: http://dblp.uni-trier.de/db/conf/dars/dars2010.html#Alonso-MoraBRBS10.
41. Berg J. van den, Lin M.C., Manocha D. Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation // IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION. — IEEE, 2008. — С. 1928—1935.
42. Aggregate Dynamics for Dense Crowd Simulation / R. Narain [и др.] // ACM Trans. Graph. — New York, NY, USA, 2009. — Дек. — Т.

28, № 5. — 122:1—122:8. — ISSN 0730-0301. — Режим доступа: http://doi.acm.org/10.1145/1618452.1618468.

43. Booth M. AI Systems of L4D / Valve. — 2009. — Режим доступа: http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_boo (дата обр. 04.08.2014).
44. Hierarchical, concurrent state machines for behavior modeling and scenario control / O. Ahmad [и др.] // Fifth Annual Conference on AI, and Planning in High Autonomy Systems. — IEEE, 1994. — С. 36—42.
45. Donikian S., Rutten É. Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation // Programming Paradigms in Graphics: Proceedings of the Eurographics Workshop in Maastricht, The Netherlands, September 2–3, 1995. — Vienna : Springer Vienna, 1995. — С. 137—153. — ISBN 978-3-7091-9457-7.
46. Badler N.I., Reich B.D., Webber B.L. Towards personalities for animated agents with reactive and planning behaviors // Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1997. — С. 43—57. — ISBN 978-3-540-68501-2.
47. Cavazza M. Al in computer games: Survey and perspectives // Virtual Reality. — 2000. — Т. 5, № 4. — С. 223—235. — ISSN 1434-9957.
48. Dromey R.G. Genetic Software Engineering – Simplifying Design Using Requirements Integration // IEEE Working Conference on Complex and Dynamic Systems Architecture. — Brisbane, 2001. — С. 1—16.
49. Dromey R.G. From requirements to design: formalizing the key steps // First International Conference onSoftware Engineering and Formal Methods, 2003.Proceedings. — Brisbane : IEEE, 2003. — С. 2—11.
50. Dromey R.G. Formalizing the Transition from Requirements to Design // Mathematical Frameworks for Component Software. — WORLD SCIENTIFIC, 2006. — С. 173—205.
51. Dromey R. Principles for Engineering Large-Scale Software-Intensive Systems // Slide Presentation for the 17th Australian Conference on Software Engineering. — 2007.

52. Towards a unified behavior trees framework for robot control / M. A. [и др.] // IEEE International Conference on Robotics and Automation (ICRA). — IEEE, 2014. — С. 5420—5427.
53. Champandard A. Understanding behavior trees. — 2007. — Режим доступа: http://aigamedev.com/open/article/bt-overview/ (дата обр. 01.12.2016).
54. Champandard A. Behavior trees for next-gen game AI. — 2008. — Режим доступа: https://aigamedev.com/insider/presentations/behavior-trees/ (дата обр. 01.12.2016).
55. Unreal Engine: Behavior Tree Quick Start Guide / Epic Games. — Режим доступа: https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/QuickStart/ (дата обр. 18.06.2016).
56. Rain AI / Rival Theory. — Режим доступа: http://rivaltheory.com/rain/ (дата обр. 18.06.2016).
57. Ogren P. Increasing modularity of uav control systems using computer game behavior trees // AIAA Guidance, Navigation, and Control Conference. — American Institute of Aeronautics, Astronautics, 2012.
58. Colledanchise M., Marzinotto A., Ögren P. Performance analysis of stochastic behavior trees // 2014 IEEE International Conference on Robotics and Automation (ICRA). — IEEE, 2014. — С. 3265—3272.
59. Dynamic Expansion of Behaviour Trees / F.-P. Gonzalo [и др.] // Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference. — AIIDE, 2008.
60. Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution / M. Perez Diego and Nicolau, M. O'Neil, A. Brabazon // Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, April 27-29, 2011, Proceedings, Part I / под ред. S. Di Chio Cecilia and Cagnoni [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. — С. 123—132. — ISBN 978-3-642-20525-5.
61. Orkin J. Agent Architecture Considerations for Real-Time Planning in Games // Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference / под ред. R.M. Young, J.E. Laird. — AAAI Press, 2005. — С. 105—110. — ISBN 1-57735-235-1.

62. Orkin J. Symbolic representation of game world state: Toward real-time planning in games // AAAI Workshop on Challenges in Game Artificial Intelligence. — AAAI Press, 2005.
63. Fikes R.E., Nilsson N.J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving // Proceedings of the 2Nd International Joint Conference on Artificial Intelligence. — London, England : Morgan Kaufmann Publishers Inc., 1971. — С. 608—620. — (IJCAI'71). — Режим доступа: http://dl.acm.org/citation.cfm?id=1622876.1622939.
64. Weber B.G., Mateas M., Jhala A. Applying Goal-Driven Autonomy to StarCraft // Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. — Stanford, California, USA : AAAI Press, 2010. — С. 101—106. — (AIIDE'10).
65. Sacerdoti E.D. The Nonlinear Nature of Plans // Proceedings of the 4th International Joint Conference on Artificial Intelligence. Т. 1. — Tblisi, USSR : Morgan Kaufmann Publishers Inc., 1975. — С. 206—214. — (IJCAI'75).
66. Tate A. Generating Project Networks // Proceedings of the 5th International Joint Conference on Artificial Intelligence. Т. 2. — Cambridge, USA : Morgan Kaufmann Publishers Inc., 1977. — С. 888—893. — (IJCAI'77).
67. Hoang H., Lee-Urban S., Muñoz-Avila H. Hierarchical Plan Representations for Encoding Strategic Game AI // Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. — Marina del Rey, California : AAAI Press, 2005. — С. 63—68. — (AIIDE'05).
68. Champandard A.J. A-Life, Emergent AI and S.T.A.L.K.E.R.: An Interview with Dmitriy Iassenev // AiGameDev.com. —. — Режим доступа: https://aigamedev.com/open/interviews/stalker-alife/ (дата обр. 01.12.2016).
69. Kelly J.-P., Botea A., Koenig S. Offline Planning with Hierarchical Task Networks in Video Games // Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. — Stanford, California : AAAI Press, 2008. — С. 60—65. — (AIIDE'08).
70. Johnston J. HTN and Behaviour Trees for improved coaching AI in RTS games // Game Behaviour. — 2014. — Т. 1, № 1.
71. Schadd M.P. Single-player Monte-Carlo tree search for SameGame // Knowledge-Based Systems. Т. 34. A Special Issue on Artificial Intelligence in Computer Games: AICG. — Elsevier, 2012. — С. 3—11.

72. AUV local path planning based on virtual potential field / D. Fu-guang [и др.] // IEEE International Conference Mechatronics and Automation, 2005. Т. 4. — 2005. — С. 1711—1716.
73. Warren C.W. Global path planning using artificial potential fields // Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on. — IEEE, 1989. — С. 316—321.
74. Warren C.W. Multiple robot path coordination using artificial potential fields // Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on. — IEEE, 1990. — С. 500—505.
75. Reif J.H., Wang H. Social potential fields: A distributed behavioral control for autonomous robots // Robotics and Autonomous Systems. — 1999. — Т. 27, № 3. — С. 171—194.
76. Conner D.C., Rizzi A.A., Choset H. Composition of local potential functions for global robot control and navigation // Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on. Т. 4. — IEEE, 2003. — С. 3546—3551.
77. Hinchey M.G., Sterritt R., Rouff C. Swarms and Swarm Intelligence // Computer. — 2007. — Т. 40, № 4. — С. 111—113. — ISSN 0018-9162.
78. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence. From Natural to Artificial Systems. — Oxford University Press, 1999. — 322 с. — (Santa Fe Institute Studies on the Sciences of Complexity). — ISBN 0195131584.
79. Madey G., Freeh V., Tynan R. Agent-based modeling of open source using swarm // AMCIS 2002 Proceedings. — 2002.
80. Chellapilla K., Fogel D.B. Evolution, neural networks, games, and intelligence // Proceedings of the IEEE. — 1999. — Т. 87, № 9. — С. 1471—1496.
81. Tan C.T., Cheng H.-l. A combined tactical and strategic hierarchical learning framework in multi-agent games // Proceedings of the 2008 ACM SIGGRAPH symposium on Video games. — ACM, 2008. — С. 115—122.
82. Tan C.T., Cheng H.-l. TAP: An Effective Personality Representation for Inter-Agent Adaptation in Games. // AIIDE. — 2008.
83. Mastering the game of Go with deep neural networks and tree search / D. Silver [и др.] // Nature. — 2016. — Т. 529. — С. 484—489.

84. Chang Y.-T., Chen S.-C., Li T.-Y. A computer-aided system for narrative creation // Computational Intelligence for Creativity and Affective Computing (CICAC), 2013 IEEE Symposium on. — IEEE, 2013. — С. 40—47.
85. Maher M., Merrki K., Sunders R. Achieving Creative Behavior Using Curious Learning Agents // AAAI Spring Symposium: Creative Intelligent Systems. — AAAI Press, 2008. — С. 40—46. — ISBN 978-1-57735-359-1.
86. Feng S., Tan A.-H. Self-organizing neural networks for behavior modeling in games // The 2010 International Joint Conference on Neural Networks (IJCNN). — IEEE, 2010. — С. 1—8.
87. Charles D., McGlinchey S. The past, present and future of artificial neural networks in digital games // Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton. — 2004. — С. 163—169.
88. Wanitchaikit S., Tangamchit P., Maneewarn T. Self-organizing approach for robot's behavior imitation // Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. — IEEE, 2006. — С. 3350—3355.
89. Environments for Multiagent Systems / D. Weyns [и др.] // The Knowledge Engineering Review. — 2005. — Т. 20, № 2. — С. 127—141.
90. Weyns D., Omicini A., Odell J. Environment as a first class abstraction in multiagent systems // Autonomous agents and multi-agent systems. — 2007. — Т. 14, № 1. — С. 5—30.
91. Weyns D., Holvoet T. A formal model for situated multi-agent systems // Fundamenta Informaticae. — Amsterdam, 2004. — Т. 63, № 2—3. — С. 125—158. — ISSN 0169-2968.
92. Modeling dynamic environments in multi-agent simulation / A. Helleboogh [и др.] // Autonomous Agents and Multi-Agent Systems. — 2007. — Т. 14, № 1. — С. 87—116. — ISSN 1573-7454.
93. Bel-Enguix G., Jimenez-Lopez M.D. Agent-environment Interaction in a Multi-agent System: A Formal Model // Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation. — London, United Kingdom : ACM, 2007. — С. 2607—2612. — (GECCO '07). — ISBN 978-1-59593-698-1.

94. Каляев И.А., Гайдук А.Р., Капустян С.Г. Модели и алгоритмы коллективного управления в группах роботов. — Москва : ФИЗМАТЛИТ, 2009. — 280 с. — ISBN 978-5-9221-1141-6.
95. Merrill B. Building Utility Decisions into Your Existing Behavior Tree // Game AI Pro. A collected wisdom of game AI professionals. — A K Peters/CRC Press, 2014. — С. 127—136. — ISBN 978-1466565968.
96. Orkin J. Three states and a plan: the AI of FEAR // Game Developer's Conference Proceedings. — GDC, 2006.
97. Naboulsi Z. Code Metrics – Maintainability Index. — Режим доступа: https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/.

Список рисунков

Список таблиц

Приложение А. Свидетельство о регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

25 пр/12
САПР
№ 7827

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012616826

Инструментарий разработчика интерактивных приложений и компьютерных игр Isilme SDK

Правообладатель(ли): ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Волгоградский государственный технический университет» (RU)***

Автор(ы): ***Алимов Александр Александрович, Шабалина Ольга Аркадьевна (RU)***

Заявка № **2012614583**
Дата поступления **5 июня 2012 г.**
Зарегистрировано в Реестре программ для ЭВМ
31 июля 2012 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

Приложение Б. Акт внедрения

Общество с ограниченной ответственностью
«МАС»
400040, г. Волгоград, ул. Руднева, 2Д, тел.+79023639186
ИНН/КПП 3442123251/344201001 ОГРН 1123459006846
Р/с № 40702810510000018599 в Ф.ЗАО АКБ «ЭКСПРЕСС-ВОЛГА» в г.ВОЛГОГРАД, БИК:041806835

Исх. № 20
от «01» сентября 2016 г.

А К Т

о внедрении научных и практических результатов диссертации на тему «Управление поведением многозадачных интеллектуальных агентов в системах реального времении»

Алимова Александра Александровича

Настоящий акт составлен о том, что в компании ООО «МАС» используются результаты диссертационного исследования Алимова А.А. «**Управление поведением многозадачных интеллектуальных агентов в системах реального времени**». На основе предложенных в диссертации архитектуры мультиагентной системы (МАС) и модели поведения агента разработано программное обеспечение, предназначенное для поддержки процесса разработки мультиагентных систем. Программное обеспечение представляет собой фреймворк для платформы .Net/Mono.

Результаты исследования, изложенные в диссертации, имеют научное и практическое значение. В результате использования моделей и алгоритмов, предложенных в диссертационном исследовании, удалось добиться повышения качества разрабатываемых мультиагентных систем.

Исполнительный директор
ООО «МАС» Алгунова Г.Е.