МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Г. В. ПЛЕХАНОВА»

Высшая школа кибертехнологий, математики и статистики
Кафедра цифровой экономики института развития информационного общества

«Допустить к защите»

Заведующий кафедрой
Цифровой экономики
института развития
информационного общества
Уринцов Аркадий
Ильич

_____
(подпись)

«____» _____ 202_ г.

Выпускная квалификационная работа бакалавра

Направление «Математическое обеспечение и администрирование информационных систем»
Профиль «Системное и интернет-программирование»

Разработка алгоритма для управления группой объектов для преследования цели в играх

Выполнил студент Федоров Кирилл Константинович

Группа 15.11Д-МО12/19б

Научный руководитель
выпускной квалификационной работы

_____
(ФИО, звание, степень, должность)

_____
(подпись)

Автор _____
(подпись)

Москва – 2023

Содержание

Оглавление

Введение

Глава I

1.1 Введение в предметную область

Алгоритмы преследования используются в разных областях жизни. Опираясь на диссертацию Ляпина Н.А., искусственный интеллект, а конкретно разные разновидности алгоритмом преследования широко используются при разработке вооружения, таких как БПЛА, БЛА или ПВО. Так, в основу БЛА закладывается алгоритм патрулирования обозначенной территории и уничтожение цели при ее обнаружении. В свою очередь, в ВПК при разработке разного вида вооружения так же используется противоположный алгоритму преследования – алгоритм уклонения. Он используется при разработке разного вида снарядов и ракет в качестве обхода ПВО, которое в свою очередь использует алгоритм преследования.

Так же, алгоритмы преследования используются в геоинформационных системах (ГИС). Они предназначены для управления большим количеством разномасштабной картографической информации, анализа взаимосвязей объектов в пространстве, управления атрибутными характеристиками объектов. Все моделируемые в ГИС объекты и явления имеют пространственную привязку, позволяющую анализировать их во взаимосвязи с другими пространственно-определенными объектами. Модули пространственного анализа и принятия решения средствами ГИС являются ядром геоинформационных технологий. На основе ГИС создаются комплексные программные решения для поддержки объектов инфраструктуры в течение их жизненного цикла: при проектировании, создании и эксплуатации транспортных сетей, энергоснабжения. ГИС уже доказали свою эффективность при поддержке эксплуатации систем энергоснабжения на электротранспорте, в частности, на городском электротранспорте и на железнодорожном транспорте.

Алгоритмов преследования и уклонения нашли широкое применение в разработке компьютерных игр. В настоящее время искусственный интеллект

широко применяется при разработке игр. Игровой искусственный интеллект — это набор программных методов, которые используют в видеоиграх для создания иллюзии разума у NPC (non-player character) через поведение персонажей. Игровой ИИ включает в себя алгоритмы теории управления, робототехники, компьютерной графики и информатики в целом. Зарождение искусственного интеллекта в видеоиграх началось до того, как сама индустрия стала неотъемлемой частью жизни практически каждого человека. Один из наиболее ранних и громких прецедентов использования этой технологии в игре относится к 1950-м годам.

Алгоритмы преследования и уклонения являются базовыми алгоритмами при проектировании центрального процессора принятия решения у игровых объектов в различных игровых жанрах. В основном они используются как одно из состояний конечного автомата игрового ИИ, означающее преследование заданной цели. Например, данная группа алгоритмов используется для ботов-патрульных, которые охраняют определённую территорию и если игрок заходит в поле зрение, то данные боты становятся ищейками, которые будут преследовать игрока. Однако, классический алгоритм преследования не решает множество проблем, связанных с преследованием цели – например, обход полигонов или непроходимых препятствий на карте.

В данной главе будут продемонстрированы различные алгоритмы преследования, способы решения проблемы обхода полигонов в определённой местности и математические модели написания искусственного интеллекта для управления группой объектов. Так же, в этой главе будут выбран и детально проанализирован оптимальный набор алгоритмический и математических моделей для решения поставленной задачи - разработка алгоритма, управляющего группой объектов для преследования цели.

1.2 Анализ математических и алгоритмических методов

Алгоритмы преследования

Метод погони

Этот метод ещё называют чистым преследованием, преследованием по кривой погони. Методом погони называется метод преследования, при котором вектор ПЕ*Vпе в любой момент времени направлен на цель, то есть его курсовой угол α равен нулю. Проанализировав данный метод, можно сделать вывод, что при методе погони, преследователь приближается к цели сзади и кривая погони характеризуется большой кривизной, при любых начальных условиях. Таким образом, даже в условиях отсутствия каких-либо маневров со стороны цели, это приводит к низкой точности преследования, что является недостатком метода. Достоинством этого метода преследования является его помехоустойчивость – для реализации метода в каждый момент времени надо знать только, слева или справа от вектора преследования находится цель, для корректировки курсового угла соответствующим образом.

Метод постоянного угла упреждения

Это метод, при котором курсовой угол в любой момент времени равен некоторой фиксированной величине α0. Величина α0 должна подчиняться условию (Vпе /Vпо)*|sinα0| < 1, в противном случае преследователь начнет описывать вокруг цели бесконечную спираль, так её и не достигнув. Этот метод является модификацией метода погони, но у него есть достоинство в том, что при использовании угла упреждения кривая погони гораздо менее искривлена, чем для метода погони. Кроме этого, метод обладает похожей помехоустойчивостью, что и метод погони. Однако для реализации этого метода необходима информация о пеленге цели, а также о направлении движения цели для выбора правильного угла упреждения.

Метод параллельного сближения

Параллельным сближением называется вид преследования, когда линия визирования всегда смещается параллельно самой себе. Если цель движется

прямолинейно и равномерно, то траектория преследователя есть прямая. При маневрах цели, когда цель получает ускорение, для сохранения условия параллельности линии визирования, ускорение преследователя будет совпадать с нормальным ускорением цели. Это является достоинством данного метода. К недостаткам метода можно отнести большое количество требуемой информации: курсовой угол цели, скорость.

Метод пропорционального наведения

У него нет тех недостатков, которые есть у метода параллельного сближения. Это такой метод, при котором угловая скорость преследователя пропорциональна угловой скорости линии визирования. Назначение этого метода – поражение цели, с учетом тенденции поворота линии визирования. Как следует из принципов действия этого метода, для его реализации необходима лишь информация о пеленге цели.

Оптимальные алгоритмы преследования

Оптимальными называются алгоритмы, который дают возможность получить оптимальное решение задачи управления с четко определенным функционалом (целевой функцией). Решение задач оптимального управления усложняется, при усложнении движения целевого объекта. Когда цель движется равномерно и прямолинейно, возможно аналитическое решение. Что невозможно в случае, когда цель маневрирует, используя сложные пространственные траектории. Подобные задачи называются стохастическими задачами преследования. Использование теории оптимального управления в решении задач наведения является важной составляющей, часто использующейся на практике, например, в частном случае преследования цели, движущейся под острым углом к встречному курсу. В таком случае классические алгоритмы погони и постоянного угла упреждения не будут работать. Теория оптимального управления позволяет получить алгоритмы преследования, которые могут решать и задачу преследования быстродвижущихся целей. Но недостатком данных алгоритмов

является большая сложность в реализации при сложных траекториях искомого объекта.

Дифференциальные игры преследования

Одной из классических дифференциальных игр является задача преследования – уклонения, когда преследователь пытается минимизировать (а цель максимизировать) время, за которое преследователь настигнет цель.

Опираясь на источник, оптимальной стратегией является метод погони. При разных игровых моделях, а также различных управлениях, оптимальными стратегиями могут стать стратегии, реализующиеся аналогами классических алгоритмов со всеми их достоинствами и недостатками.

Учитывая то, что перед данной дипломной работой не стоит задача реализации сложных алгоритмов преследования, которые применяются в военно-промышленной области, например, учитывающие физические нюансы, классический алгоритм преследования, или алгоритм погони, наиболее подходит для поставленной цели. Однако, использование обычного метода погони без каких-либо изменений не вполне рационально, потому что предполагается, что на местности, на которой будет работать данный алгоритм, будут располагаться препятствия в виде полигонов разной величины. Опираясь на источник, использование обычного метода погони, приведет к тому, что объекты-преследователи не будут способны обходить препятствия и просто будут застревать в них, пытаясь пройти через них. К тому же, задача дипломной работы заключается в реализации алгоритма, управляющего группой объектов. Из статьи следует, что в играх в основном используется комбинированная модель поведения объектов – они управляются единым интеллектом, однако конкретные функции, например, преследование, реализуется на конкретном объекте-преследователе. По сути, единый интеллект распоряжается лишь состояниями подконтрольными объектами.

Для решения задачи обхода полигонов существует множество методов, имеющие принципиально разные концепции и разные входные данные.

Методы решения проблемы обхода полигонов

Метод с предварительной оценкой местности

Статья предлагает метод для решения проблемы обхода полигонов, основывающийся на предварительной оценке карты местности и последующей корректировки знаний. Карта, в свою очередь, представляется в виде двумерного массива, состоящая из единиц и нулей. Каждая ячейка этого массива представляет собой отдельную клетку, где 0 – клетка проходима, а 1 – не проходима. Так же, данный метод имеет два случая – NPC (Non-Player Character) «знают» местоположение игрока и обратный случай.

В первом, когда NPC «знаю» местоположение игрока, ситуация в понятиях игрового искусственного интеллекта называется cheat-методом. Игрок ставится в неравное положение по отношению к неигровым персонажам, и тем самым повышается сложность игры. Проведенный предварительный анализ проблемы показал, что проблема преследования обычно решается двумя основными способами:

- использование алгоритмов поиска пути;
- комбинация алгоритмов поиска пути и алгоритмов преследования.

С точки зрения практической реализации, первый способ является самым простым. При изменении местоположения игрока пути до него просчитываются заново. Подобный подход имеет серьезные недостатки, связанные с большими временными затратами, а также с большим потреблением памяти.

Если в качестве алгоритма поиска пути выбран алгоритм A*, то временные затраты, в худшем случае, растут экспоненциально, аналогично экспоненциально растут затраты памяти.

Второй способ более сложный. Путь до игрока вычисляется с помощью алгоритмов поиска пути. Если расстояние между игроком и неигровым

персонажем становится меньше R, то используется более простой и менее ресурсоемкий алгоритм преследования. Эффективность такого метода определяется, прежде всего, величиной радиуса видимости R и размером карты.

NPC «не знаю» о местоположении игрока. Методы из данной группы добавляют в игру больший реализм. Данные методы требуют, чтобы неигровые персонажи были в состоянии предсказывать местоположение игрока. В простейшем случае они могут просто двигаться в случайных направлениях. Как только игрок попадает в область видимости неигрового персонажа, дальнейшее преследование осуществляется при помощи соответствующих алгоритмов. Таким образом, первоочередной задачей становится задача оценки вероятности появления игрока в той или иной ячейке карты.

К разработанному алгоритму на этапе проектирования были выдвинуты следующие требования:

- результатом его работы должна быть матрица весов, каждый элемент которой–вес конкретной ячейки игровой карты;
- алгоритм должен обеспечивать учет предыдущих игровых исходов, т.е. быть самообучающимся;
- первоначальный расчет должен проводиться с учетом положения входа и выхода и структуры карты. Таким образом, в функционировании алгоритма можно выделить три этапа:
- этап инициализации;

  На данном этапе проводится инициализация матрицы весов
- этап обучения;

  Вычисления на данном этапе происходят в конце каждой игровой сессии.
- — этап корректировок.

В ходе работы алгоритма возникают ситуации, когда необходимо нормировать значения в матрице весов.

Описанный метод был реализован в компьютерной игре Paclight. На произвольных картах размерности 40 на 40 метод показал свою пригодность. Проведенные наблюдения показали, что на картах такого размера время, затраченное на расчет путей, было меньше примерно в1,2разапосравнению с применение только A* (в случаях, когда NPC знают местоположение игрока). Ожидается, что с увеличением размерности карт выгода от применения метода предварительной оценки карт возрастет.

Хранение информации о местности

Наиболее простой способ обойти проблему столкновения объектов-преследователей с полигонами состоит в том, чтобы сохранить некоторую "информацию об окружении" (ИО) внутри каждого объекта. Это означает, что если объект утыкается в твердый объект, то он мог бы "спросить" этот объект, куда двигаться, чтобы обойти его.

В зависимости от стороны, с которой произошло столкновение, каждое препятствие возвращает значения пары смещения dx/dy относительно объекта. Правила, для нахождения этих значений:

- Допустим, объект свободно перемещается по экрану, к примеру, из левой верхней к правой нижней его части. И он соприкоснулся с левой стороной препятствия, тогда пускай он движется вниз.
- Если в препятствие ударились по направлению какой-нибудь оси, возвратим противоположное направление по этой же оси. Если препятствие имеет ИО (-1, 1) и получает удар от чего-то движущегося по оси X, следует возвратить 1.

Очевидный недостаток этого приема в том, чтобы сохранить другие два значения для каждого объекта. Возможным решением данной проблемы станет сохранение данной информации, используя всего лишь 4 бита.

Движение объекта-преследователя по выпускаемым лучам

Идея данного алгоритма в выпускании от каждого бота преследователя 2 отрезка сенсора, направленные на игрока. При пересечении полигона лучом, данный луч должен поворачиваться (один поворачивается по часовой стрелке, другой против) до тех пор, пока они не перестанут пересекать полигоны и далее объект-преследователь движется по тому лучу, который быстрее добрался до преследуемой цели и раньше перестал пересекать полигоны.

Однако, у этой идеи есть свои недостатки. Во-первых, она достаточно трудоемкая в силу того, что на каждой итерации надо проверять от каждого объекта-преследователя лучи пересекаются ли они с каждым ребром полигона, которых потенциально может быть много на местности. Во-вторых, данный алгоритм не всегда приводит к ожидаемым результатам. Если длина луча слишком коротка, то бот может зайти в тупик, из которого придется выбираться.

Например:

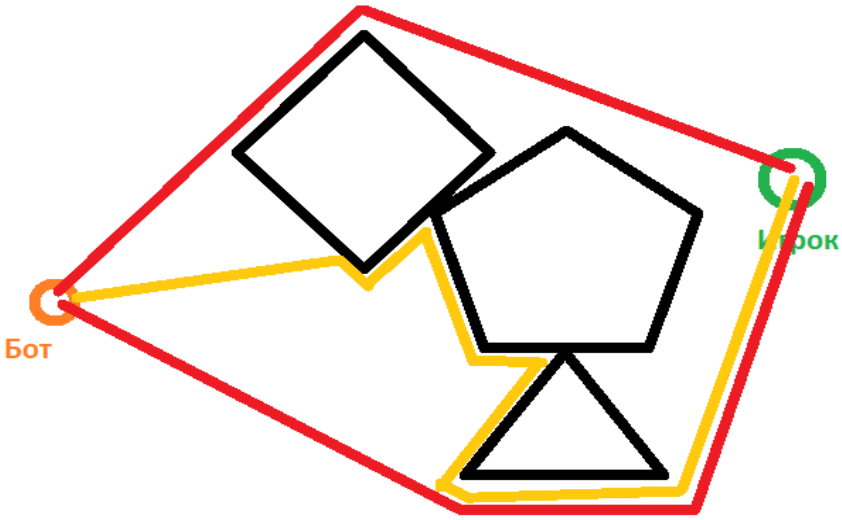

Рисунок 1.

На Рисунок 1, бот пойдет оранжевым путем, когда красный явно оптимальнее.

Однако, если длина луча слишком длинная, то бот, наоборот, примет узкий проходит за тупит и будет обходить его, как на Рисунок 2.

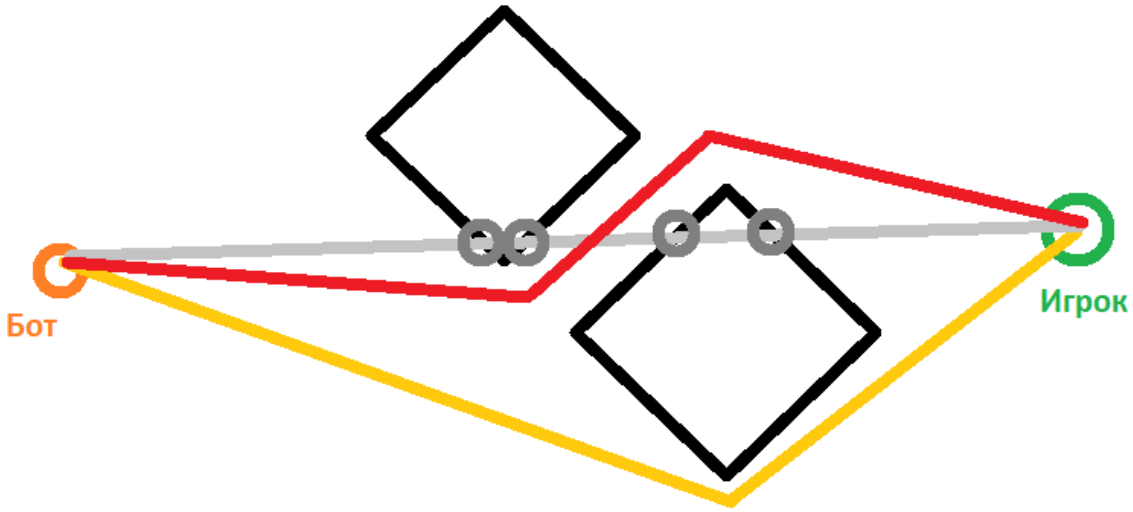

Рисунок 2.

Серый вектор на Рисунок 2 – луч, пересекший полигоны. Здесь бот пойдет по оранжевому пути, хотя красный оптимальнее.

Частично эти проблемы нивелируются, если игрока, объектов-преследователей и вершины полигонов представить в виде вершин графа и совершать поиск наикратчайшего пути, например при помощи алгоритма Дейкстры.

Однако, проблема оптимизации остается прежней. Чем больше карта, полигонов и объектов-преследователей, тем медленнее работает алгоритм, потому что для каждой итерации придется пересчитывать алгоритм Дейкстры на большое дерево и для каждого объекта преследователя.

Метод ищейки

Данный метод основывается на том, что преследуемый объект оставляет следы запаха. Объекты-преследователи, когда не видят преследуемый объект, стараются ориентироваться по оставляемым следам как ищейки. В основу метода закладывается конечный автомат состояний в объект-преследователя. Когда объект видит цель, он просто старается сближаться с ней при помощи классического алгоритма преследования. Когда объект-преследователь не видит цели, то его состояние переключается на ориентирование по оставляемым следам.

Этот метод имеет ряд преимуществ. Во-первых, он не ограничен размерами карты, ее формой, количеством полигонов и формой полигонов. Во-вторых, данный алгоритм энергоемкий по времени и по памяти, опираясь на статью, описывающая данный алгоритм.

Исходя из требования, что алгоритм должен быть максимально возможно гибким и подходить для любого типа местности и полигонов, то алгоритм ищейки наиболее подходит для данной цели, так как остальные алгоритмы имеют ограничения по поводу размера и типа местности. К тому же, он наиболее ресурсоемкий, так остальные алгоритмы либо требуют много времени на начальной инициализации, либо требуют очень много ресурсов в процессе работы.

Групповое мышление объектов

Ранее были рассмотрены алгоритмы и математические модели, применяемые для одного конкретного объекта. Задача данной работы разработать алгоритм, управляющий объектами-преследователями. По сути, задача разработать искусственный интеллект, который в зависимости от игровой ситуации будет менять поведение подконтрольных объектов. Искусственный интеллект нужен для имитации разумности NPC, при этом его задача не в том, чтобы обыграть пользователя, а в том, чтобы развлечь его. В современных играх используются разные подходы для создания ИИ. Опираясь на статью, в основе лежит общий принцип: получение информации → анализ → действие.

Получение информации происходит примерно так же, как и в реальном мире — у ИИ есть специальные сенсоры, при помощи которых он исследует окружение и следит за происходящим. Сенсоры бывают совершенно разными. Это может быть традиционный конус зрения, «уши», которые улавливают громкие звуки, или даже обонятельные рецепторы. Конечно, такие сенсоры — всего лишь имитация реальных органов чувств, которая позволяет сделать игровые ситуации более правдоподобными и интересными. Наличие и

реализация сенсоров зависит от геймплея (игровой процесс). Во многих активных шутерах (жанр игр, основанный на высокой концентрации боев) не нужны комплексные рецепторы — достаточно конуса зрения, чтобы реагировать на появление игрока. А в стелс-экшенах (жанр игр, основанный на «бесшумном» прохождении игры) весь геймплей основан на том, чтобы прятаться от противников, поэтому виртуальные органы чувств устроены сложнее.

Когда ИИ получил информацию, он начинает «обдумывать» свои действия, анализируя обстановку. Обычно в этом участвует сразу несколько систем ИИ, отвечающих за разные вещи. Часто разработчики добавляют подобие коллективного интеллекта, который следит за тем, чтобы действия отдельных агентов не противоречили и не мешали друг другу. При этом сами мобы зачастую даже не знают о существовании своих союзников — эта информация им не нужна, потому что за координирование действий отвечает ИИ более высокого уровня.

В играх есть несколько подходов, которые чаще всего используются для принятия решения. Один из самых простых и понятных подходов — это rule-based ИИ. В основе лежит список правил и условий, заранее созданный разработчиками. Такой подход можно эффективно использовать для создания простого поведения. Например, «если игрок приближается к курице ближе, чем на три метра, то она начинает от него убегать».

Следующий распространённый способ принятия решений — конечные автоматы (КА, finite state machine, FSM). Этот подход позволяет NPC беспроблемно переходить между разными состояниями. Например, есть моб, базовое состояние которого — патрулирование по определённой траектории. Если внезапно появится игрок, NPC перейдёт в новое состояние — начнёт стрелять. Конечные автоматы как раз обеспечивают эти переходы: они принимают информацию с предыдущего состояния и передают в новое.

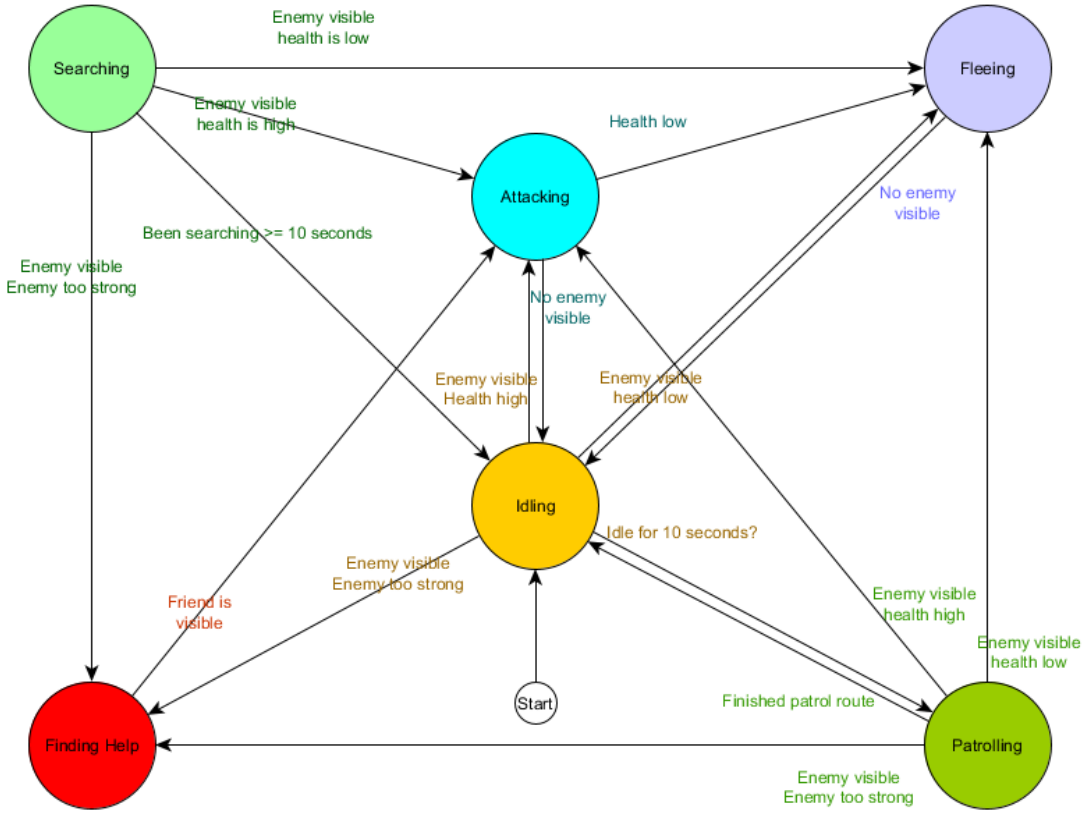

Рисунок 3. Пример конечного автомата принятия решения

Преимущество этого подхода в том, что персонаж всегда будет находиться в каком-то состоянии и не зависнет где-то между ними. Так как разработчик должен прописать все переходы, он точно знает, в каких состояниях может находиться игровой объект. Недостаток метода в том, что с увеличением количества механик значительно разрастается и система конечных автоматов. Это увеличивает риск появления ошибок, а также может снизить скорость операций.

Дерево поведения — это более формализованный подход построения поведения мобов. Его особенность заключается в том, что все состояния персонажа организованы в виде ветвящейся структуры с понятной иерархией. Дерево поведения содержит в себе все возможные состояния, в которых может оказаться моб. Когда в игре происходит какое-то событие, ИИ проверяет, в каких условиях находится NPC, и перебирает все состояния в поисках того, которое подойдёт для нынешней ситуации.

Дерево поведения отлично подходит для того, чтобы систематизировать состояния NPC в играх, в которых есть множество механик и игровых элементов. В ситуации, когда моб участвует в перестрелке, ему не нужно будет искать подходящее действие в ветке патрулирования. Такой подход помогает сделать поведение NPC отзывчивым и обеспечивает плавный переход между разными состояниями.

Иерархические конечные автоматы объединяют особенности конечных автоматов и дерева поведения. Особенность такого подхода в том, что разные графы внутри логики могут отсылаться друг к другу. Например, нам надо прописать поведение для нескольких мобов. Не обязательно делать для каждого отдельную логику — можно создать общее базовое поведение и просто отсылаться к нему при необходимости.

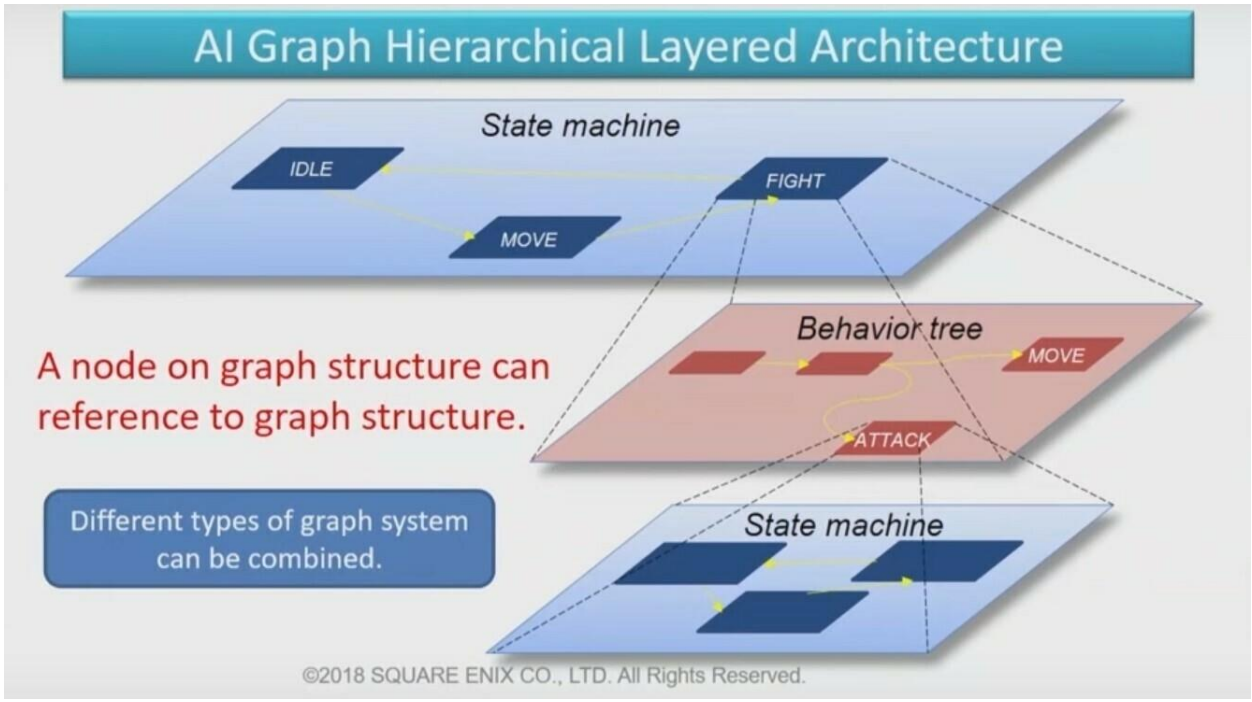

Рисунок 4. Иерархическая архитектура в Final Fantasy XV.

Есть и менее популярные решения, которые не смогли полноценно закрепиться в индустрии. К примеру, в F.E.A.R. использовалась система целеориентированного планирования действий (Goal-Oriented Action Planning, GOAP) — для всех NPC она создаёт план действий, основываясь на

информации об игровом мире. Например, если мобу нужно перейти в другую комнату, то система сперва проверяет, какое расстояние нужно пройти до двери, есть ли на пути препятствия, открыта ли дверь и так далее. Когда у системы есть вся информация об окружении, она составляет план, а NPC просто проигрывает последовательность анимаций.
Этот подход работает на основе конечных автоматов, но они отвечают только за воспроизведение анимаций. У автоматов есть всего три состояния, каждое из которых отвечает за свой набор анимаций: движение (бег, ходьба), действия (стрельба, реакции), взаимодействие с объектами (открыть дверь, включить свет).

Очевидно, что выбор определенной модели поведения объектов зависит от целей и задач, которые они должны выполнять. Данная работа преследует цель создать групповой, алгоритм, который будет способен управлять подконтрольными объектами для преследования цели. В общем случае цепочка принятия решения сводится к обнаружению объекта любым из подконтрольных объектов и началу движения за ним всех остальных подконтрольных объектов. Если объект-жертва стоит на месте, то задача группового алгоритма сводится к окружению данной жертвы так, чтобы она не могла выйти из окружения. Для этой цели подходит модель конечного автомата, так как всего есть три состояния: состояние спокойствия, состояние преследования, состояние окружения. Они будут переключаться в зависимости от игровой ситуации.

1.3 Постановка задачи

В данном параграфе будет сформулирована задача разработки, описана математическая и алгоритмическая модель и озвучены требования к техническим характеристикам, а также проанализированы аналоги, представленные на рынке.

Прежде, чем переходить к вышеперечисленным планам, стоит рассмотреть существующие на рынке ПО (программное обеспечение),

реализующие групповой алгоритм для преследования цели. Так как данный алгоритм в рамках данной выпускной квалификационной работы выполняется в качестве отдельного модуля к существующему ПО для разработки игр (в дальнейшем называемые «Игровые движки»), то стоит рассмотреть существующие на рынке в открытом доступе аналогичные ПО, реализующие похожую технологию.

Аналоги на рынке

Первым игровым движком к рассмотрению будет Unity Engine. Это кроссплатформенная среда разработки компьютерных игр, созданная компанией Unity Technologies. Так как задачей данной ВКР заключается в создании простого искусственного интеллекте, который будет управлять подконтрольными объектами для преследования цели, то стоит рассматривать уже готовые AI (artificial intelligence) модели в игровых движках, в Unity они называются «machine learning agents". Полагаясь на официальную статью от компании Unity Technologies, внутри данного игрового движка заложены специальные объекты, которые способны обучаться тем задачам, которые в них закладывает разработчик. По сути, это шаблон, который нужно обучать для каких то конкретных задач. То есть, разработчики данного игрового движка не закладывали готовый функционал алгоритма, который будет детально изучаться в данной ВКР.

Вторым для рассмотрения игровым движком будет не менее известный Unreal Engine, который так же находится в открытом доступе, созданный компанией Epic Games. Из официальной документации ясно, что игровой движок предлагает несколько решений для создания собственного искусственного интеллекта. Конкретно, внутри данного движка уже заложен функционал для создания дерева поведения и автомата состояний, которые рассматривались в параграфе *Групповое мышление объектов*. Так же, данный игровой движок предлагает готовые решения, реализующие навигационную систему в собственной системе координат. Однако готового решения,

предлагающее групповой алгоритм, управляющий подконтрольными объектами для преследования цели, в данном игровом движке нет.

Задача разработки

После анализа существующих аналогов на рынке можно сделать вывод, что явной реализации технологии, управляющей группой объектов для преследования цели нет. Задача данной ВКР разработать готовый модуль для собственного игрового движка, которая будет реализовывать данную технологию.

Основная задача – сделать так, чтобы данный модуль работал с максимально возможной эффективностью и на любом типе местности.

После анализа математических и алгоритмических методов в параграфе **1.2 Анализ математических и алгоритмических методов** были выбраны конкретные методы для решения определённых задач. По сути, разработка алгоритма, управляющего группой объектов для преследования цели, сводится к 3 задач: выбор метода преследования, выбор метода реализации группового алгоритма и решение проблемы обхода полигонов на местности.

Так, было установлено, что наилучшим методом преследования цели будет метод погони или классический метод преследования, который характеризуется своей простатой и эффективностью.

Для реализации группового алгоритма будет использоваться модель, основанная на использование конечных автоматов, так как это позволит четко контролировать состояния подконтрольных объектов и самого алгоритма, обеспечивая предсказуемость поведения алгоритма, что гарантирует точность его работы. К тому же автоматы состояний обеспечивают эффективность работы алгоритма.

Для решения проблемы обхода полигонов будут использоваться два разных алгоритма, которые будут сменять друг друга в зависимости от состояния конкретного объекта. В первом случае будет использоваться метод ищейки. Использование данного метода обуславливается его гибкостью и

эффективностью. Гибкость обеспечивается за счет того, что данный метод не привязан к локальной системе координат. Он способен работать на любом типе местности с любым количеством непроходимых полигонов. Эффективность же обуславливается, что данный метод построен на автоматах состояний, которые переключается в зависимости от текущей ситуации и компьютеру не придется долго принимать решение. Так же, данный метод, в отличие от остальных, не принуждает к хранению значительного количества информации, такую как сетку местности, координаты конкретных полигонов и т. д., внутри одного игрового объекта. Таким образом, данный легко работает на любом типе местности и при ее изменении так же будет работать без дополнительных изменений внутри самого алгоритма или дополнительной информации.

Во втором случае будет использоваться алгоритм нахождения наикратчайшего пути. Данный метод не обладает конкретными преимуществами и эффективность его спора, однако он необходим в редких игровых ситуациях, так что его использование обязательно и сильно нагружать систему он не будет.

Описание поведения математической модели

Описание поведения мат модели сводится к работе конечного автомата состояний группового алгоритма и подконтрольных алгоритмов. Состояния подконтрольных объектов будут и состояния управляющего ими объекта, реализующий алгоритм группового управления, взаимозависимы. Поэтому, описания автоматов состояния будут происходить параллельно.

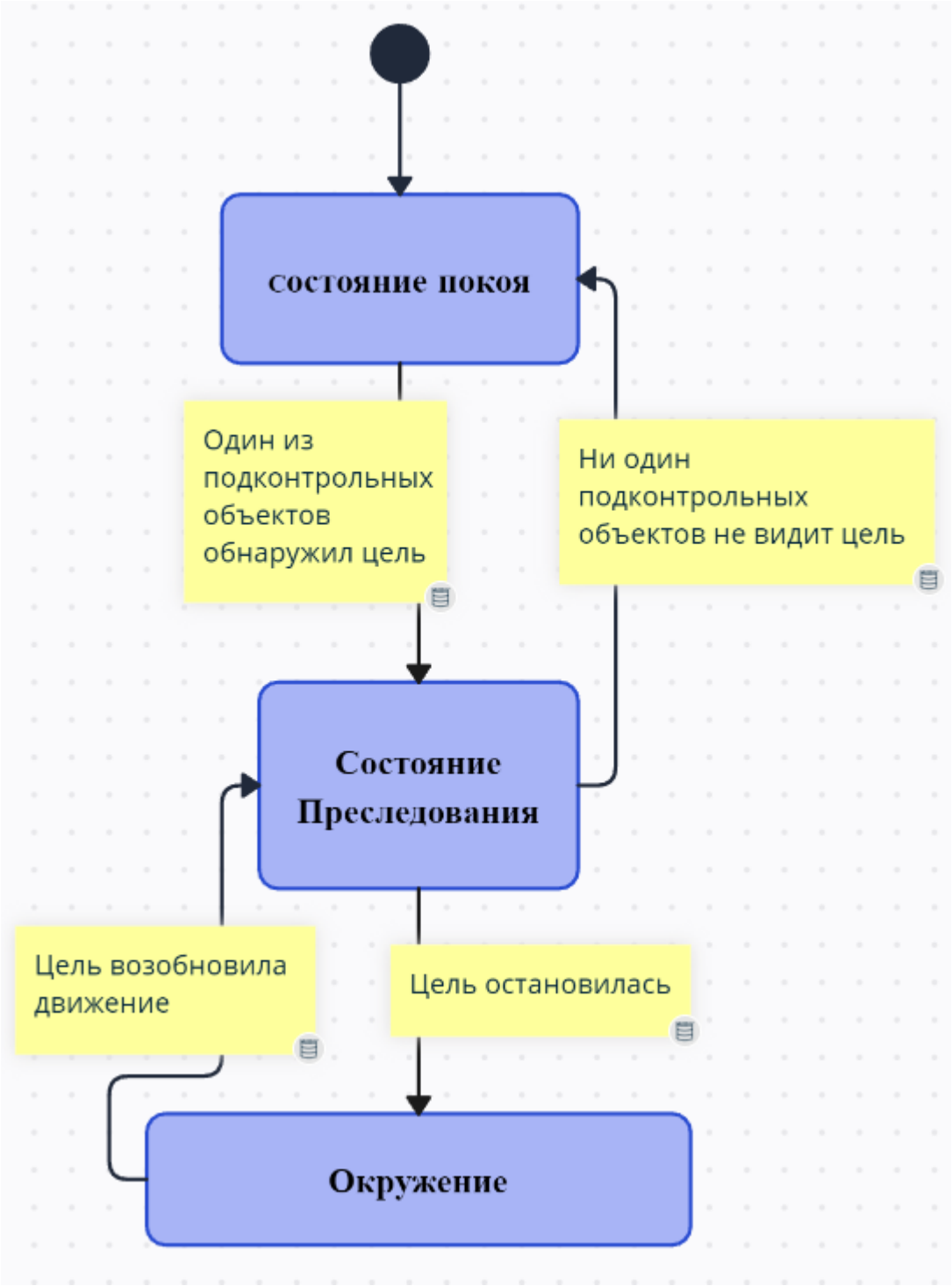

Рисунок 5. Автомат состояний группового алгоритма

Как видно из Рисунок 5, всего существует 3 возможных состояния группового алгоритма.

Первое состояние — это состояние покоя. С начала работы программы алгоритм будет находится именно в этом состоянии. Оно характеризуется тем,

что все подконтрольные объекты не выполняют никаких задач, а сам алгоритм не распределяет никаких задач между подконтрольными объектами.

Второе состояние – это состояние преследования. В него групповой алгоритм может перейти, только если один из подконтрольных объектов «замечает» цель и переходит в состояние преследования, используя метод ищейки, то есть двигаясь по «следам» цели, как показано на Рисунок 6.

Данное состояние характеризуется тем, что групповой алгоритм оповещает все подконтрольные ему объекты о смене состояния покоя на состояние преследование и начинает передавать координаты преследуемой цели тем объектам, которые не двигаются по следам, оставляемые целью.

Групповой алгоритм выходит из этого состояние в случаях, когда:

- Ни один из подконтрольных объектов не видит преследуемую цель
- Цель остановилась

В первом случае групповой алгоритм переходит в состояние покоя и оповещает подконтрольных объектов об этом. Во втором случае алгоритм переходит в состояние окружения. В этом состоянии у группового алгоритма задача окружить преследуемый объект. Алгоритм старается расставить подконтрольные объекты равномерно вокруг цели.

Если преследуемая цель возобновляет движение, то групповой алгоритм обратно переход в состояние преследования и оповещает об этом подконтрольные объекты.

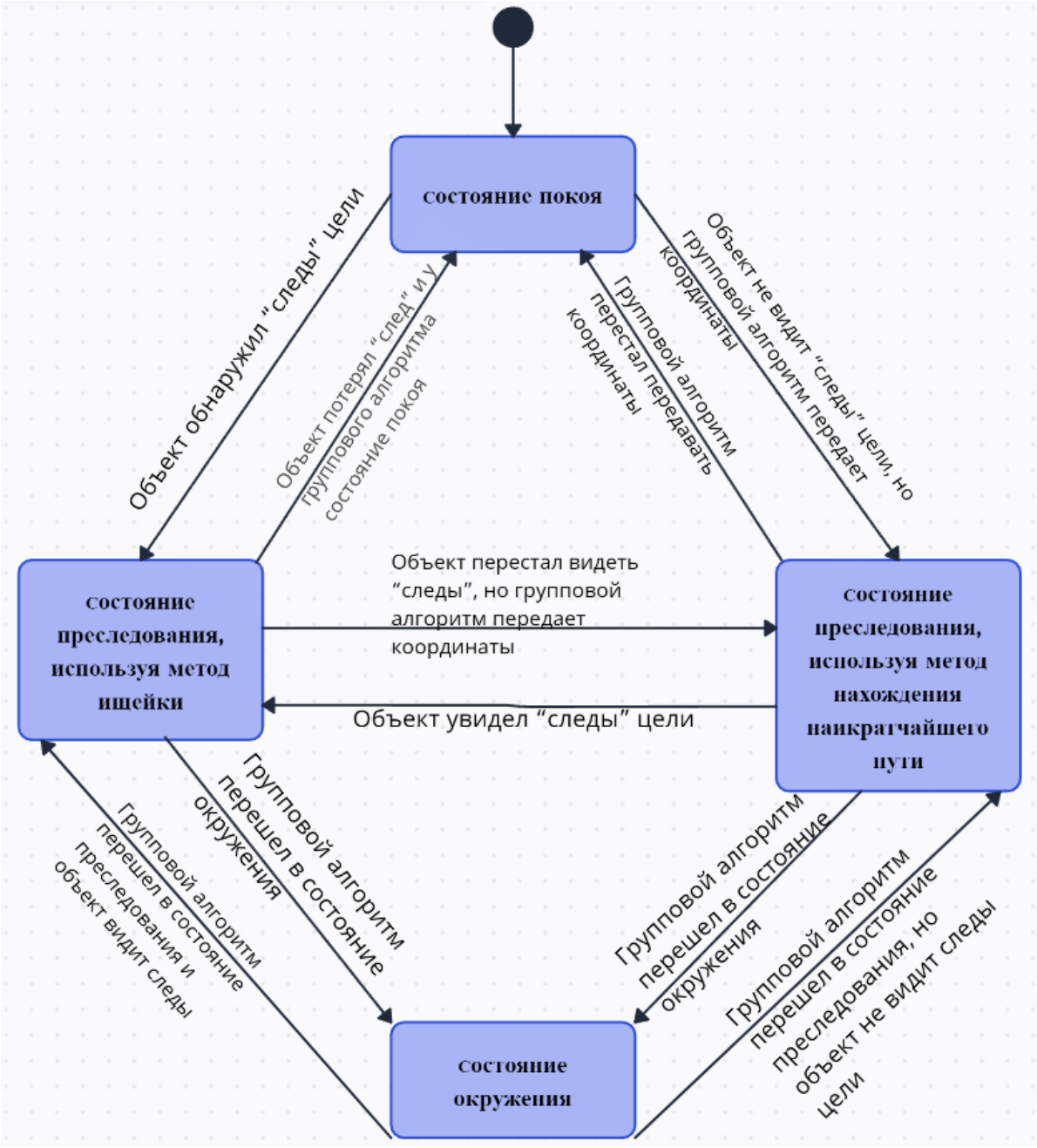

Рисунок 6. Автомат состояний управляемого объекта

На Рисунок 6 продемонстрирован автомат состояний объектов-преследователей или подконтрольного объекта. Он аналогичен автомату состояний группового алгоритма, однако состояние преследования разделена на два разных состояния.

Первым состоянием является состояние покоя. В него входят все объекты-преследователи на этапе инициализации.

Если один из подконтрольных объектов замечает цель, которую надо преследовать, он переходит в состояние преследования и оповещает об объект, который производит контроль над данным, передавая ему координаты цели. Данный объект оповещает все остальные подконтрольные ему объекты, переводя их в состояние преследования, отсылая им координаты цели. Таким образом, часть объектов-преследователей находится в состоянии преследования, используя метод ищейки, то есть двигаясь по следам, оставляемые целью, а другая часть старается наикратчайшем путем дойти до координат, которые им передает групповой алгоритм. То есть, задача главного алгоритма перевести все ему подконтрольные объекты в состояние преследования по следам.

Подконтрольные объекты и групповой алгоритм находятся в состоянии преследования, пока хотя бы один из подконтрольных объектов двигается по следам цели. Как только все подконтрольные объекты переходят в состояние преследования по координатам, групповой алгоритм переходит в состояние покоя, переводя все подконтрольные объекты в состояние покоя.

Если преследуемый объект останавливается, то групповой алгоритм переводит все подконтрольные ему объекты в состояние окружения, стараясь «окружить» цель. Если преследуемый объект возобновляет движение, то групповой алгоритм переходит в состояние преследования, переводя подконтрольные объекты в одно из двух состояний преследования:

- Если преследуемый объект видит следы, то он переходит в состояние преследования, используя метод ищейки
- Если преследуемый объект не видит следы, то он переходит в состояние преследования, стараясь сократить расстояние с преследуемой целью наикратчайшим путем

Таким образом, групповой алгоритм и подконтрольные ему объекты работают в синергии, переключая друг друга состояния в зависимости от игровой ситуации, стараясь наиболее эффективно догнать преследуемую цель.

Список литературы

1. https://cyberleninka.ru/article/n/sravnenie-algoritmov-presledovaniya-obektov/viewer
2. https://programcpp.ucoz.ru/publ/4-1-0-6
3. https://intuit.ru/studies/courses/1104/251/lecture/6456
4. https://cyberleninka.ru/article/n/problema-presledovaniya-v-igrah-snizhenie-resursoemkosti-resheniya-s-pomoschyu-metoda-s-predvaritelnoy-otsenkoy-kart/viewer
5. https://qna.habr.com/q/870001
6. https://habr.com/ru/post/496878/
7. https://habr.com/ru/company/netologyru/blog/598489/
8. https://habr.com/ru/post/420219/
9. https://www.dissercat.com/content/razrabotka-i-issledovanie-algoritma-garantiruyushchego-upravleniya-traektoriei-bespilotnogo
10. https://unity.com/products/machine-learning-agents

11. https://docs.unrealengine.com/5.0/en-US/artificial-intelligence-in-unreal-engine/
12.