Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем
Направление подготовки 09.04.04 Программная инженерия
Направленность (профиль) образовательной программы Управление разработкой программного обеспечения

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«_____»_____ 2023 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему: Применение многослойного перцептрона при разработке игровых приложений

Исполнитель
студент группы 157-ом _____ (подпись, дата) П.О. Кащеев

Руководитель
доцент, канд. техн. наук _____ (подпись, дата) Т.А. Галаган

Руководитель научного
содержания программы
магистратуры
профессор, доктор техн. наук _____ (подпись, дата) И.Е. Ерёмин

Нормоконтроль
доцент, канд. техн. наук _____ (подпись, дата) Л.В. Никифорова

Рецензент
_____ (подпись, дата) О.Г. Какаулин

Благовещенск 2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Бушманов
подпись
«_____»_____2023 г.

З А Д А Н И Е

К магистерской диссертации студента группы 157-ом

Кащеева Павла Олеговича

1. Тема магистерской диссертации: Применение многослойного перцептрона при разработке игровых приложений

(Утверждено приказом от 21.02.2023 № 442-уч)

2. Срок сдачи студентом законченной работы (проекта): 20.06.2023

3. Исходные данные к магистерской диссертации: Предметная область, проектная документация, отчёты по практической подготовке, перечень учебной литературы, интернет ресурсы.

4. Содержание магистерской диссертации: анализ предметной области; поиск существующих решений; моделирование объекта исследования

5. Перечень материалов приложения: (наличие чертежей, таблиц, графиков, схем, программных продуктов, иллюстративного материала и т.п.): UML схемы

6. Рецензент магистерской диссертации: О.Г. Какаулин

7. Дата выдачи задания 30.01.2023

Руководитель выпускной квалификационной работы: Галаган Т.А. доцент кафедры ИиУС, канд. тех. наук, доцент

Задание принял к исполнению (30.01.2023): _____

РЕФЕРАТ

Магистерская диссертация содержит 64 страницы, 7 рисунков, 1 приложение, 52 источника.

ИГРОВОЕ ПРИЛОЖЕНИЕ, ПРОЕКТИРОВАНИЕ, АЛГОРИТМ, ИГРОВОЙ ОБЪЕКТ, ТЕСТИРОВАНИЕ, МНОГОСЛОЙНЫЙ ПЕРЦЕПТРОН, НЕЙРОННЫЕ СЕТИ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Цель работы – создание игрового приложения с использованием модели поведения игрового персонажа при помощи многослойного перцептрона.

В работе исследованы существующие решения данной задачи. Описаны основные аспекты проектирования и разработки системы и приложения. Была выбрана наиболее подходящая для реализации и соответствующая требованиям среда. Описаны классы, их процедурные модули и атрибуты. Также было проведено тестирование работоспособности проекта.

Задачи работы:

– изучение возможности применения ИИ в игровых приложениях;
– определение функционала игрового приложения;
– формулирование требований к программному продукту;
– проектирование программного продукта и взаимодействия его модулей;
– проектирование структуры нейронной сети и её применение в модели поведения игрового персонажа;
– реализация нейронной сети;
– разработка, отладка и тестирование игрового приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир невозможно представить без информационных технологий. С каждым годом компьютерные технологии становятся всё более востребованными и развиваются всё сильнее. Они находят применение во многих сферах жизни, от повседневного пользования телефоном до решения глобальных проблем человечества, таких как наука, экология, медицина и т.д. Более того, на текущий момент сфера информационных технологий является активно возрастающим и развивающимся сектором экономики. Такое успешное и активное развитие отрасли можно объяснить как доступностью самих компьютерных технологий, так и их большими вычислительными мощностями и универсальностью в выполнении большинства задач.

Одной из наиболее актуальных областей в развитии информационных технологий на сегодняшний день является область искусственного интеллекта. Главной задачей данной области является воссоздание виртуальной модели нервной системы человека и протекающих в ней процессов, а также использование данной модели при решении тех или иных задач. Существует множество методов решения данной задачи. Одним из таких методов является использование многослойного перцептрона – нейронной сети, способной обучаться и прогнозировать результаты на основе имеющихся данных.

Данная технология всё чаще находит применение в различных сферах. Одной из возможных областей применения данной технологии является область разработки игр, а точнее разработки искусственного интеллекта для игровых оппонентов. Данная тема актуальна, поскольку использование нейронных сетей в качестве модели поведения противника позволит воспроизвести схожее с человеческим поведение неких виртуальных объектов в той или иной смоделированной ситуации. Также применение данного метода в разработке игровых приложений может повысить их качество и улучшить пользовательский опыт.

В данной магистерской диссертации рассматривается процесс разработки игрового приложения с использованием многослойного перцептрона. В работе

рассматривается актуальность технологий искусственного интеллекта, возможность их применения при разработке игрового приложения, а также существующие концепции и реализации игрового искусственного интеллекта.

Работа содержит описание этапов разработки приложения. Дано описание существующих и планируемых функциональных и архитектурных особенностей приложения. Также представлены результаты тестирования и анализа эффективности применения данного метода.

В данной работе описывается алгоритм работы игрового программного приложения и нейросети. Также описываются возможности и причины выбора программно–технического обеспечения, с помощью которого разработан данный программный продукт.

1 ПРИМЕНЕНИЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ИГРОВЫХ ПРИЛОЖЕНИЯХ

1.1 Предметная область темы исследования

1.1.1 Искусственный интеллект

Искусственный интеллект – это научное направление, сутью которого является решение задачи аппаратного и программного моделирования интеллектуальной человеческой деятельности[1]. Другими словами, это область, направленная на создание интеллектуальных систем, способных решать различные задачи так, как их мог бы решить человек. При этом некоторые методы, которыми достигается данная цель, не всегда могут быть точной копией работы человеческого мозга.

Как научное направление искусственный интеллект тесно связан с другими науками, например, с психологией, биологией и лингвистикой. Специалисты в этих областях строят и программно реализуют, совместно с математиками, всё новые и новые модели. На их основе исследователи в области ИИ пытаются восстановить конкретные формы проявления интеллекта. Полученные при этом результаты, в свою очередь, дают новую информацию к размышлению для специалистов данных областей.

Создание искусственного интеллекта основывается на применении различных подходов, каждый из которых использует различные принципы и технологии разработки. В данной сфере существует множество подходов, с помощью которых можно реализовать данную систему. И единственного верного и универсального подхода нет. Это обусловлено как тем фактом, что у науки до сих пор нет чёткого ответа на вопрос, чем занимается или должен заниматься искусственный интеллект, так и тем, что каждый из подходов находит успешное применение при решении разных задач.

Несмотря на наличие множества подходов к пониманию задач искусственного интеллекта и созданию интеллектуальных информационных систем, можно выделить два основных типа подходов к разработке искусственного ин-

теллекта – «нисходящий» или семиотический (англ. Top-Down AI) и «восходящий» или биологический (англ. Bottom-Up AI) [2]. Главная задача данных подходов – полное или частичное воспроизведение принципа работы человеческого мозга во время интеллектуальной деятельности.

Принцип «нисходящего» подхода заключается в проектировании высокоинтеллектуальной системы на основе некоторой уже существующей информации. Примером таких систем являются экспертные и логические системы или базы знаний, имитирующие различные высокоуровневые психические процессы. Знания в данных системах формализуются в виде неких продукционных импликационных правил, иерархических структур (фреймы), семантических сетей или в виде аппарата алгебры логики для описания и интерпретации знаний (предикаты первого порядка). На основе одной из форм создаются механизмы интерпретации этих знаний (алгоритм рождается в процессе работы). В результате получаются системы декларативного типа, в которых знания сосредоточены не в программе, а в информационных структурах, сама программа – только универсальный решатель.

Принцип «восходящего» подхода подразумевает под собой проектирование системы с помощью моделирования механизмов мышления человека. При данном подходе система обучается алгоритму работы не с помощью базы формальных знаний, а на основе внутренних запрограммированных процессов.

1.1.2 Подходы к разработке искусственного интеллекта

Как было описано ранее, в сфере искусственного интеллекта существует множество различных подходов к его реализации. Одним из таких подходов является нейронная сеть. Нейронная сеть (также искусственная нейронная сеть, ИНС) – математическая модель, состоящая из множества искусственных нейронов, представляющих собой простые обработчики сигналов [3]. Данный подход относится к восходящим подходам проектирования искусственного интеллекта. Понятие нейронной сети возникло при изучении процессов, протекающих в мозге и при попытке смоделировать эти процессы. После разработки алгоритмов обучения получаемые модели стали использовать в практических

целях: в задачах прогнозирования, для распознавания образов, в задачах управления и др.

ИНС представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Данные нейроны состоят из множества входных и выходных каналов. Основная часть нейрона представляет собой взвешенный сумматор, который получает по входным каналам сигналы x1…xn, а затем определяет выходной сигнал через функцию суммы f(u) этих входных сигналов. (см. Рисунок 1). Данная функция называется функцией активации.

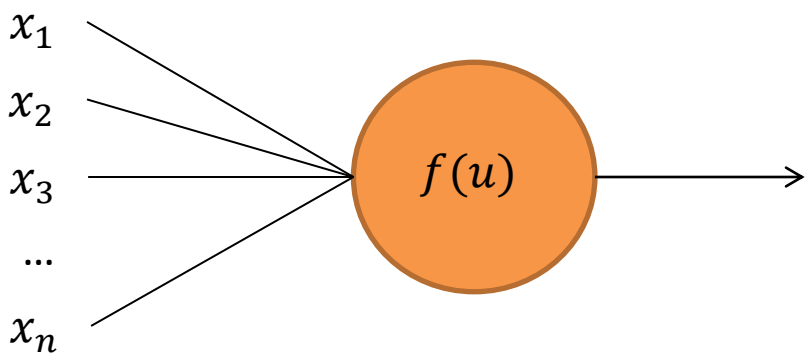

Рисунок 1 – Общая схема математического нейрона

Каждый нейрон подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим нейронам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые сумматоры вместе способны выполнять довольно сложные задачи.

Существует множество различных архитектур нейронных сетей. Каждая из них отличается по структуре и характеристикам. Перцептронами называют нейронные сети прямого распространения, т.е. входной сигнал в таких сетях распространяется в прямом направлении, от слоя к слою[4]. Перцептрон в общем представлении состоит из следующих элементов:

– множества входных узлов, которые образуют входной слой;
– одного или нескольких скрытых слоев вычислительных нейронов;
– одного выходного слоя нейронов.

Все нейроны связаны между собой синоптическими связями, каждая из которых имеет вес – силу связи. Количество входных и выходных элементов в перцептроне определяется условиями задачи. Также ими определяется и количество вычислительных слоёв. Многослойными перцептронами называются перцептроны с 2 и более внутренними слоями.

Перцептроны имеют следующих три отличительных признака:

– каждый нейрон сети имеет нелинейную функцию активации. Важно подчеркнуть, что такая нелинейная функция должна быть гладкой (т.е. всюду дифференцируемой), в отличие от жесткой пороговой функции, используемой в перцептроне Розенблатта. Самой популярной формой функции, удовлетворяющей этому требованию, является сигмоидальная. Наличие нелинейности играет очень важную роль, так как в противном случае отображение «вход-выход» сети можно свести к обычному однослойному перцептрону;

– несколько скрытых слоев. Перцептрон содержит один или несколько слоев скрытых нейронов, не являющихся частью входа или выхода сети. Эти нейроны позволяют сети обучаться решению сложных задач, последовательно извлекая наиболее важные признаки из входной информации;

– высокая связность. Перцептрон обладает высокой степенью связности, реализуемой посредством синаптических соединений. Изменение уровня связности сети требует изменения множества синаптических соединений или их весовых коэффициентов.

Перцептроны, как и остальные архитектуры нейронных сетей, обучаются необходимому алгоритму поведения и работы. Возможность обучения − одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей

выборке, а также неполных и/или «зашумленных», частично искажённых данных.

Машинное обучение − это область сферы искусственного интеллекта, сутью которой является изучение и создание компьютерных алгоритмов, способных автоматически улучшаться благодаря предыдущему опыту и использованию внешних данных [5].

Различают два типа обучения:

– обучение по прецедентам, или индуктивное обучение, которое основано на обнаружении и идентификации закономерностей в получаемых данных;

– обучение по знаниями, или дедуктивное обучение, которое основано на анализе логики и закономерностей в некоей формализованной базе знаний экспертов.

Оба этих типа имеют общий принцип работы. Он заключается в определении закономерности между неким набором входных данных (объектов) и набором возможных ответов, причём данная закономерность существует изначально, но обучаемая система о ней не знает. Задача обучения заключается в поиске данной неявной зависимости. Благодаря этому система выстраивает алгоритм, который может для любого возможного входного объекта выдать достаточно точный классифицирующий ответ. Важным свойством обучаемой системы является обобщаемость, т.е. способность адекватно реагировать на данные за пределами существующих обучающих примеров.

Существует несколько способов обучения интеллектуальных и экспертных систем. К ним относятся обучение без учителя и с учителем. Обучение без учителя заключается в определении закономерности самой системой в ходе работы. Обучение с учителем подразумевает, что существует некоторый вспомогательный алгоритм, который непосредственно участвует в обучении интеллектуальной системы. При обучении система вознаграждается за хорошие ответы и наказывается за плохие. Они могут быть проанализированы с точки зрения теории решений, используя такие понятия как полезность.

Комбинация всех этих свойств наряду со способностью к обучению на собственном опыте обеспечивает вычислительную мощность многослойного перцептрона. Однако эти же качества являются причиной неполноты современных знаний о поведении такого рода сетей: распределенная форма нелинейности и высокая связность сети существенно усложняют теоретический анализ многослойного перцептрона.

В машинном обучении для обучения нейронных сетей прямого распространения широко используется алгоритм обратного распространения (BP). Обратное распространение ошибки − это метод, который выводит первые частные производные функции ошибки нейронной сети по весам сети. Исходя из этого, правила обучения оптимизируют сеть, корректируя веса таким образом, чтобы различия между выходами сети и соответствующим наблюдением в среднем были минимальными. Существуют обобщения обратного распространения для других искусственных нейронных сетей (ИНС) и функций в целом. При подборе нейронной сети обратное распространение эффективно вычисляет градиент функции потерь по отношению к весам сети. Благодаря этой эффективности можно использовать градиентные методы для обучения многослойных сетей и обновлять веса для минимизации потерь; обычно используется градиентный спуск или его варианты, такие как стохастический градиентный спуск. Алгоритм обратного распространения вычисляет градиент функции потерь по отношению к каждому весу, используя цепное правило. Он включает в себя вычисление градиента слой за слоем и итерацию назад от последнего слоя, чтобы избежать избыточных вычислений промежуточных членов в цепном правиле. Такие задачи относятся кдинамическому программированию.

1.1.3 Актуальность темы исследования

Область развития искусственного интеллекта является одной из наиболее актуальных областей в развитии информационных технологий на сегодняшний день. Использование различных подходов проектирования искусственного интеллекта при решении тех или иных задач позволяет автоматизировать и усовершенствовать различные рабочие процессы, которые требуют человеческих

физических и умственных затрат.

Искусственный интеллект может повысить эффективность работы благодаря возможности эффективно выполнять определенные повторяющиеся задачи без какого-либо вмешательства человека. Это не только снижает количество человеческого труда, необходимого для выполнения задач, но также устраняет возникновение человеческих ошибок. В отличие от людей, машины ИИ не устают и могут работать непрерывно в течение долгого промежутка времени. Это избавляет от необходимости нанимать более одного сотрудника для выполнения одной и той же задачи в разные рабочие смены. Человек может совершать ошибки даже при пристальном внимании и при обладании необходимыми навыками. Машины, которые работают на основе ИИ, способны решать сложные задачи и выполнять задачи с большей точностью. Точность этих машин на основе искусственного интеллекта повышает их ценность во многих отраслях.

Еще одним преимуществом ИИ в современном мире является то, как эта технология способна обрабатывать информацию. Машины на основе ИИ способны обрабатывать огромные объемы данных гораздо быстрее, чем человек. Из этого следует ещё одно преимущество машинного обучения. Оно устраняет необходимость в непрерывном обучении. Для эффективной работы человеку необходимо обучаться в течение длительного времени. Машины ИИ также учатся намного быстрее, что делает их намного более эффективными в случае обновления. Более того, при усовершенствовании рабочей техники человеку необходимо переучиваться, что также занимает время, в то время как для машин на основе ИИ требуется, чтобы машина включала изменения в способы выполнения задач. И стоит также отметить, что стоимость эксплуатации этих машин может быть намного меньше, чем их человеческие аналоги в значительной степени благодаря их высокой степени точности и эффективности.

На сегодняшний день технологии искусственного интеллекта постепенно вводятся в различные сферы. При этом области применения интеллектуальных систем достаточно узкие, то есть эти системы могут работать только в одной

конкретной области. В основном ИИ используется в таких сферах как финансы, государственное управление, военное дело и спецслужбы, медицина, тяжелая промышленность, управление человеческими ресурсами, изобразительное искусство, музыка, новости, издательство и писательство, онлайн и телефонные службы поддержки клиентов, техническое обслуживание телекоммуникаций и транспорт, развлечения и игры.

В настоящий момент в области искусственного интеллекта наблюдается вовлечение многих предметных областей, имеющих скорее практическое отношение к ИИ, а не фундаментальное. Многие подходы были опробованы, но к возникновению искусственного разума ни одна исследовательская группа пока так и не подошла. Таким образом, на данный момент развитие данной сферы зависит от постепенного совместного продвижения в различных областях, и поэтому тема, связанная с изучением возможности применения существующих технологий искусственного интеллекта в сфере игровых развлечений, также может внести небольшой вклад в дальнейшее развитие. Помимо развлекательной цели, создание модели поведения некоего игрового объекта может найти применение и в других областях, например в области моделирования различных процессов, ситуаций, различных симуляторах и тренажёрах и т.д.

1.2 Общая характеристика исследуемой задачи

1.2.1 Игровой искусственный интеллект

В современной игровой индустрии технологии искусственного интеллекта и нейросетей используются во многих направлениях. Это и улучшение графики, и оптимизация производительности игры и игровых процессов, и отображение анимации объектов, и построение или генерация уровней, и даже создание сюжетных событий. Однако одним из наиболее актуальных и при этом же наименее развитым направлением является использование нейросетей в моделировании поведения игровых оппонентов. Это очень перспективное направление, поскольку благодаря нейросетям возможно создать такую модель поведения противника, которая могла бы быть похожа на поведение персонажа, которым мог бы управлять человек. Она могла бы планировать и менять дей-

ствия, основываясь на действиях игрока и на общей игровой ситуации. Благодаря этому действия компьютерного оппонента будут менее предсказуемыми, что сделает геймплей игры разнообразнее и интереснее.

Игровой искусственный интеллект – одно из направлений ИИ. Оно подразумевает под собой разработку набора программных методик, которые используются в компьютерных играх для создания иллюзии интеллекта в поведении персонажей, управляемых компьютером. Игровой ИИ, помимо методов традиционного искусственного интеллекта, включает также алгоритмы теории управления, робототехники, компьютерной графики и информатики в целом.

Персонажей компьютерных игр, управляемых игровым искусственным интеллектом, делят на:

– боты (англ. Bot) – враждебные к игроку персонажи, приближающиеся по возможностям к игровому персонажу. В игре они противостоят игроку, и как правило, они встречаются либо по одному, либо небольшими группами. Боты наиболее сложны в программировании, поскольку при их создании должны учитываться возможные игровые ситуации и реакции игрока на действия противника.

– неигровые персонажи (англ. Non-player character, NPC) – дружественные или нейтральные к игроку персонажи. В одних случаях это могут быть просто мирные жители. Тогда их действия расписываются либо набором правил и сценариев, либо более сложными концепциями, если по механике игры их отношение к вам может меняться с дружеского на враждебное. В других случаях NPC могут быть игровыми напарниками, и тогда их поведение прописывается точно так же, как и у ботов. Единственное отличие – действия NPC должны быть направлены на защиту и помощь игроку, а также на противодействие оппонентам игрока.

Для каждого типа персонажа применяется одна из нескольких моделей поведения игрового персонажа. Существует несколько концепций, которые используются в играх. Первым из них является ИИ, построенный на наборе правил. Поведение игровых персонажей основывается на прописанных алгорит-

мах, которые конкретным образом реагируют на определённые игровые ситуации. Примером подобного поведения являются противники, которые постоянно двигаются в одном направлении и атакуют. Из-за разнообразных действий может отличаться конечный результат, но подобную систему нельзя назвать интеллектуальной. Если игрок разберётся в шаблоне, то он всегда сможет предсказывать действия.

Второй тип – ИИ на конечных автоматах. Здесь поведение организовано с помощью конечного числа состояний, между которыми переключается игровой объект. Данная концепция используется в отношении объектов, которые изменяются и меняют поведение в зависимости от определённых условий или событий. Примером данного поведения являются боты, которые находятся в состояние «патрулирование», но если игрок попадает в их поле зрения, то они автоматически переходят в состояние «нападение» [6].

Более совершенной моделью является адаптивный ИИ. Данная система используется в играх, имеющих сложную механику и разнообразие возможностей. Применяется данный тип в играх, которые необходимо сделать сложными и непредсказуемыми, и в которых предыдущие два типа ИИ будут предсказуемы. При использовании адаптивного ИИ в системе регистрируется выбор, осуществлённый игроком в ходе каких-то событий. Все переменные оцениваются, и на их основании строится будущее поведение. Оценить ситуацию можно с помощью различных параметров, например, состояния здоровья, количество времени, затраченного на определённые действия, количество ресурсов, наличие преимуществ и т. д. В играх, построенных на тактике, адаптивный ИИ может также учитывать способы победы игрока в предыдущих играх и изменять поведение, переходя от нападения в оборону, или наоборот. Адаптивный ИИ может «экономно» вести бои с минимальными потерями, выбирать сбалансированную трату войска на победу или идти в наступление не учитывая потери.

1.2.2 Использование нейросетей в качестве игрового ИИ

На данный момент нейросети в играх в качестве ИИ противника используются редко, поскольку с их разработкой возникают определённые сложности.

Трудность заключается в том, что реализация игрового ИИ сильно влияет на геймплей, системные требования и бюджет игры, и разработчики балансируют между этими требованиями, стараясь сделать интересный и нетребовательный к ресурсам ИИ малой ценой. Поэтому подход к игровому ИИ серьёзно отличается от подхода к традиционному ИИ – широко применяются разного рода упрощения, обманы и эмуляции. Например: с одной стороны, в шутерах от первого лица безошибочное движение и мгновенное прицеливание, присущее ботам, не оставляет ни единого шанса человеку, так что эти способности искусственно снижаются. С другой стороны боты должны делать засады, действовать командой и т.д., для этого применяются различные хитрости в виде сценарных и контрольных точек, расставленных на уровне.

Такое упрощение происходит, во-первых, потому что в большинстве современных игр очень много различных механик, с которыми может взаимодействовать как игрок, так и оппонент, и учесть все механики довольно проблематично. Из этого вытекает и вторая причина – компаниям невыгодно тратить трудовой ресурс на создание ИИ с помощью нейросетей, поскольку в современных играх упор больше делается на контент игры, т.е. на создание мира, сюжет, геймплейные механики и т.д. И в-третьих, при разработке ИИ очень велика вероятность создать модель с неправильными данными, которая может нарушить баланс сил в игре, из-за чего оппонент будет либо слишком слабым для игрока, либо же наоборот слишком сильным. Поэтому в современных играх нейросети в качестве искусственного интеллекта практически не применяются.

Тем не менее, смысл для применения технологий искусственного интеллекта в качестве модели поведения игрового оппонента существует. В отличие от компьютерных ботов, которые лишь запрограммированы на определенные действия или имитацию игрока, нейросеть способна мыслить, принимать решения, динамически менять обстановку по ходу игры, а значит вплотную приближается к человеку-игроку или даже превосходит его. Также нейросеть способна на обучение, а это значит, что в игре не будет одинаково повторяющихся ситуаций, которые всегда бывают с ботами. Однако это свойство переходит и в не-

достаток, когда нейросеть значительно обыгрывает живого соперника по мере роста знаний об игре.

1.3 Обзор существующих методов решения задачи

В игровой индустрии есть множество примеров реализации моделей поведения персонажей в играх. В каждой из них по-разному реализованы модели поведения, однако по большому счёту во всех играх используются алгоритмы, а не нейронные сети. Тем не менее, проекты с полным или частичным использованием технологий искусственного интеллекта также существуют. Например, Creatures – программа-симулятор искусственной жизни, в которой пользователь выращивает маленьких пушистых существ «норнов» и учит их, как себя вести. Эти существа могут разговаривать, есть, защищаться от других злых существ. Это было первое популярное приложение, реализовавшее концепцию машинного обучения в интерактивном моделировании. Существа используют нейронные сети, чтобы узнать, что им делать, а также могут обучать друг друга некоторым действиям. Игра считается прорывом в исследованиях искусственной жизни, целью которых является моделирование поведения существ, взаимодействующих с окружающей их средой.

Другим примером использования технологий искусственного интеллекта является Façade – компьютерная игра в жанрах квест и симулятор. В Façade игрок берёт на себя роль старого друга семьи Трипа и Грейс. Сюжет строится нелинейно благодаря нескольким вариантам развития событий и трём типам концовки. Персонажи реагируют на каждое действие игрока, даже на простое перемещение по квартире или направление взгляда. Цель протагониста – примирить семейную пару, которая очень долгое время жила счастливо, пока не случился серьёзный скандал, или, наоборот, рассорить её.

Нейросети используются и в некоторых стратегиях в качестве некоторого вспомогательного механизма. Так, в Planetary Annihilation разработчики используют искусственные нейронные сети в своем агенте искусственного интеллекта по умолчанию. А в Supreme Commander 2 используются многослойные перцептроны для управления реакцией взвода на встреченные вражеские под-

разделения. Всего используется четыре перцептрона, по одному для каждого типа взвода: сухопутный, морской, бомбардировщик и истребитель.

Также технологии ИИ используются и в качестве системы построения сюжета. AI Dungeon – это текстовая приключенческая игра, в которой искусственный интеллект используется для создания неограниченного сюжета в ответ на действие игрока. В игре предлагается выбрать сеттинг для своего приключения (например, фэнтези или научная фантастика), за которыми следуют другие параметры, относящиеся к сеттингу. Игрок с помощью команд, которые можно выбрать для каждого фрагмента текста, может взаимодействовать с персонажами, совершать действия или сам корректировать историю. Игра адаптируется и реагирует на большинство команд и действий, вводимых игроком. Пустой ввод также можно использовать, чтобы побудить ИИ к генерации дальнейшего контента. Кроме того, игра предоставляет игрокам возможность отменить/повторить или изменить недавние события, чтобы улучшить повествование игры. Игрокам также предоставляется возможность явно указать ИИ, какие элементы следует «запомнить» для использования в дальнейшем прохождении.

В вышеописанных проектах нейронные сети используются по большей части как механизм управления игровым процессом. В качестве оппонента игровой искусственный интеллект встречается как отдельная система. Наиболее яркие примеры:

Deep Blue – ИИ, разработанный IBM для игры в шахматы. Победил чемпиона мира по шахматам. Матч Каспаров против супер ЭВМ не принёс удовлетворения ни компьютерщикам, ни шахматистам, и система не была признана Каспаровым.

AlphaGo – система, разработанная Google DeepMind. Предназначена для игры в го. Выиграла матч против корейского профессионала 9 дана Ли Седоля. Впоследствии была улучшена до более сильной версии AlphaGo Zero. Новая нейросеть обучалась играть в го с нуля, играя сама с собой (без обучения на партиях, сыгранных людьми, как это было в более ранних версиях AlphaGo).

AlphaStar – программа, играющая в стратегическую игру StarCraft II, также разработанная компанией DeepMind. Данная программа стала первым ИИ, достигнувшим уровня лучших игроков в киберспортивной дисциплине при отсутствии специальных ограничений. В конце 2018 года был проведён ряд тестовых матчей против профессиональных игроков, закончившийся чистой победой ИИ, а в 2019 году AlphaStar принял участие в рейтинговом режиме StarCraft II, в результате чего достиг высшего внутриигрового ранга лиги – грандмастера.

Watson – перспективная разработка IBM, способная воспринимать человеческую речь и производить вероятностный поиск, с применением большого количества алгоритмов. Для демонстрации работы Watson принял участие в американской игре «Jeopardy!», аналога «Своей игры» в России, где системе удалось выиграть в обеих играх.

2 АЛГОРИТМИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ РЕШЕНИЯ ЗАДАЧИ

2.1 Предлагаемый алгоритм решения задачи

Проект состоит из двух постоянно взаимодействующих между собой частей: непосредственно сама игра и нейросеть. Задача нейросети заключается в управлении игровым оппонентом, в то время как задача игры заключается в отображении как изменения состояния этого оппонента, так и изменения состояний игрока и игрового поля.

2.1.1 Структура игры

В разрабатываемой игре перед игроком ставится цель пройти все уровни, каждый из которых представляет собой лабиринт из нескольких связанных друг с другом комнат. Комнаты представляют собой двухмерную плоскость, по которой игрок может передвигаться во всех направлениях. Сами комнаты состоят из стен, дверей, которые открываются ключами или просто нажатием кнопки взаимодействия, предметов, которые могут помочь игроку в прохождении, и противников.

Для работы игры необходимы файлы, в которых записаны информация об уровне и сохранённая конфигурация приложения, а также ввод данных с клавиатуры и мыши. В ответ пользователю в игровом окне будет выводиться изображение игрового процесса, и по окончании игры также будет выводиться результат игры. Диаграмма работы приложения представлена на рисунке 2.

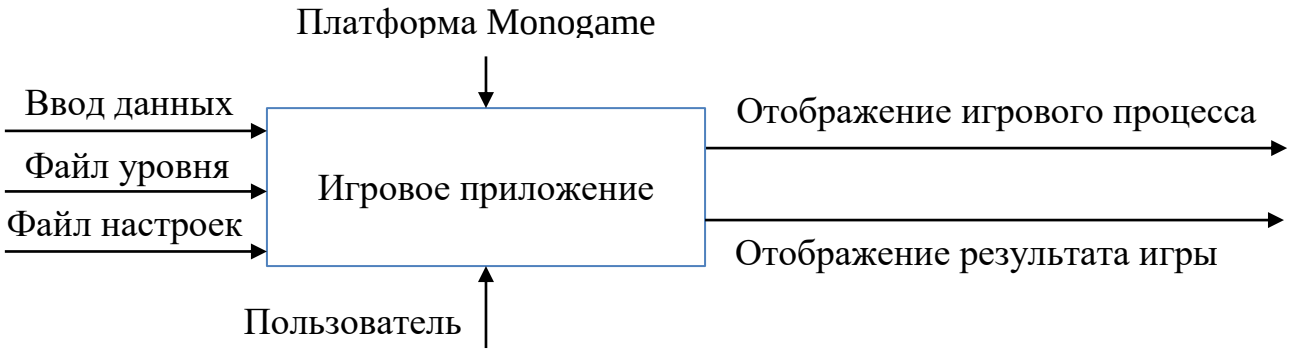

Рисунок 2 – Диаграмма работы игрового приложения

Игра начинается со стартовой позиции, которая указывается в файле уровня. Игрок начинает игру с полным количеством очков здоровья и начальным оружием. Очки здоровья он может потерять при столкновении с противником или с поверхностями, которые могут нанести вред. Сама игра представляет собой набор из нескольких последовательных уровней, по которым перемещается игрок. Каждый уровень представляет собой двухмерный массив из комнат, а каждая комната - двухмерный массив из плиток, являющихся проходимыми для игрока и противников (пол) или непроходимыми (стена). Также каждая комната содержит в себе определённое количество противников и предметов.

И противники, и игрок в данной игре обладают некоторыми одинаковыми параметрами – скоростью перемещения по игровому полю, здоровьем, поэтому для описания каждого из них используется некоторая абстрактная сущность. При этом учитывается тот факт, что и у игрока, и у противников есть свои характеристики, которые необходимо описывать отдельно. Также игроку предоставляется оружие, которым он может защищаться от противников. Для удобства пользователя предоставляется графический дисплей (HUD, т.е. HeadUpDisplay), который показывает текущее состояние игрока.

В процессе пользования игрового приложения участвует всего один человек – пользователь (или игрок). Его основными действиями в игре являются инициация игры, участие в игровом процессе, сохранение и загрузка текущего игрового прогресса, настройка параметров игры, просмотр справки и выход.

Инициация игры происходит при нажатии пользователем одной из кнопок – «Начать игру» или «Загрузить игру», расположенных в главном меню. В первом случае происходит запуск игры с самого начала, а во втором – с некоторой контрольной точки сохранённого прогресса. Но и в том, и в другом случае происходит инициация игры, то есть загрузка игрового персонажа, игровых оппонентов и объектов.

Процесс игры подразумевает под собой активное взаимодействие пользователя и окружающего игрового мира. Пользователь действует при помощи игрового персонажа, которым он управляет. Персонаж может передвигаться по

игровому полю, атаковать противников, взаимодействовать с предметами и открывать двери. При этом игровой процесс не останавливается, если пользователь не предпринимает никаких действий. Сам уровень продолжает обновляться до тех пор, пока персонаж жив. Если персонаж погибает в ходе игры, то ему будет предложено запустить игру с момента последнего сохранения или просто выйти из игры. Также игрок в любой момент может выйти из игры сам.

В случае если игрок не дошёл до конца игры, но на данный момент ему необходимо завершить игру, он может сохранить прогресс через меню паузы, или при нажатии на специальную кнопку на клавиатуре. При следующем запуске игры он может либо начать новую, либо запустить игру с того момента, на котором остановился.

Если игроку требуется помощь в некоторых аспектах игры, в главном меню он может найти кнопку справки. Нажав на неё, он переходит в подменю, в котором находится информация об игровых противниках, некоторых предметах, об управлении игроком и т.д.

Для более удобной игры в меню предусмотрено подменю Настройки, с которым пользователь может взаимодействовать. В этом окне пользователь может выбрать настройки графики, звука, управления и игрового процесса. Полная схема вариантов использования представлена в приложении А.1.

Проект состоит из 6 модулей – модуль игрового интерфейса, модуль загрузки уровня, модуль игры, модуль взаимодействия игровых объектов и модуль игрового интерфейса. Основным модулем является модуль игры. Задача этого модуля – создание и обеспечение как работы игрового цикла (принцип работы которого будет описан далее), так и взаимодействия программных модулей. Взаимосвязь между всеми основными модулями представлена на рисунке 3.

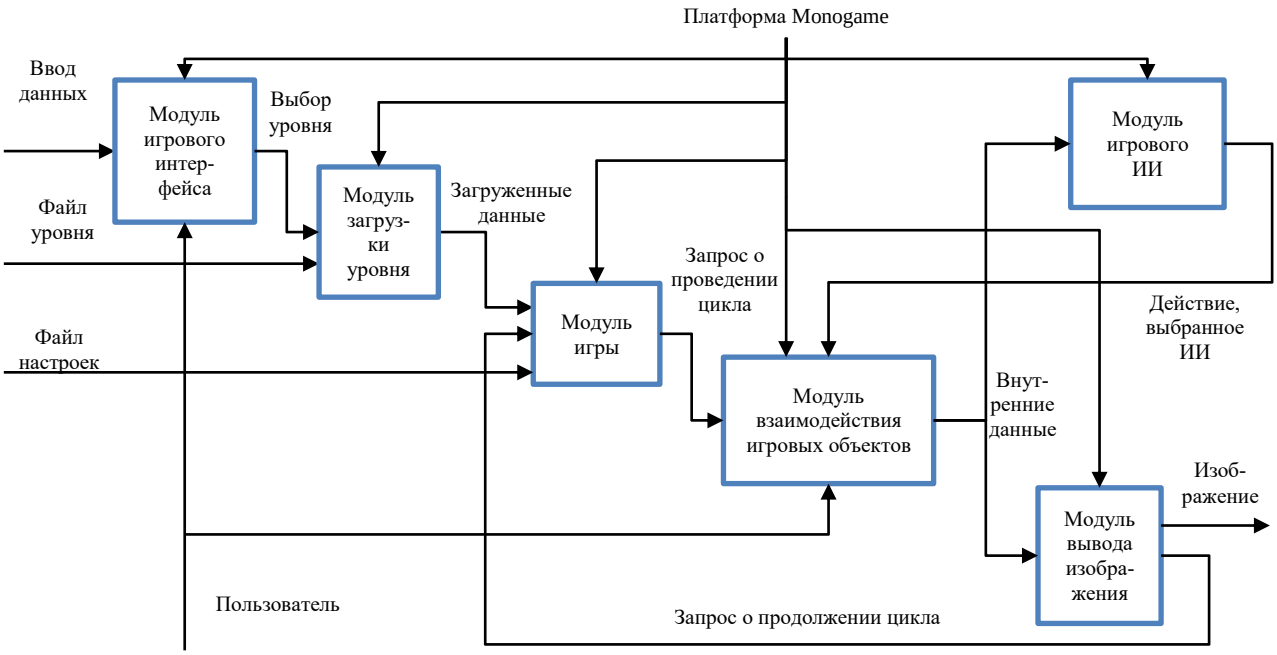

Рисунок 3 – Декомпозиция процесса работы приложения

Модуль игрового интерфейса представляет собой внутриигровое меню и является вспомогательным. Данный модуль предназначен для взаимодействия игрока с игровым приложением. Благодаря ему пользователь может запустить игровой процесс, настроить его или завершить работу приложения. Через этот модуль происходит вызов модуля игры. При нажатии на кнопку «Начать» или «Загрузить», пользователь передаёт сигнал этому модулю о том, что требуется загрузка определённого уровня. После этого блок игры обращается к модулю загрузки уровня. У этого блока основная задача – считывание данных уровня из файла. Если у пользователя уже есть некоторый прогресс, то этот модуль дополнительно считывает файлы, хранящие прогресс игры. После завершения блок посылает ответный сигнал обратно блоку игры.

Следующий этап работы программы – игровой цикл. Данный цикл состоит из взаимодействия модуля игры с модулем взаимодействия игровых объектов и модулем вывода изображения. Сначала блок игры отправляет сигнал блоку игровых объектов. Этот блок считывает ввод игрока, и затем обновляет все внутриигровые объекты. Часть обработанных данных он отправляет в модуль искусственного интеллекта (принцип работы данного модуля описан в следующем разделе), после чего получает от этого модуля ответ в виде действия, ко-

торое совершит управляемый перцептроном игровой противник. Далее уже на основе этого действия модуль обновляет остальные данные и затем отправляет сигнал о завершении обратно в блок игры. Далее по полученным в ходе обновления данным с помощью блока вывода изображения происходит отрисовка игрового поля, объектов, игрока и пользовательского интерфейса с характеристиками персонажа. После отрисовки сигнал возвращается обратно к блоку игры, он снова посылает сигнал модулю взаимодействия игровых объектов, и так происходит до тех пор, пока игра не будет окончена. Если игрок сохраняет свой прогресс, то сигнал от блока игры получает блок уровня. Если игрок выходит из игры, то сигнал получает блок меню. Полная диаграмма последовательности представлена в приложении А.2.

Игровое приложение может быть в нескольких состояниях. Первое состояние, в которое программа попадает при запуске – ожидание ввода. Пользователь попадает в главное меню игры, и программа ждёт от пользователя нажатия кнопки меню (ввода). При старте игры программа переходит в состояние загрузки уровня. В этом состоянии она только показывает процесс загрузки уровня. После загрузки начинается состояние игрового процесса. В этом состоянии приложение активно взаимодействует с пользователем. В любой момент времени пользователь может открыть меню паузы, тем самым останавливая игровой цикл и переводя систему в состояние паузы и ожидания ввода. Отсюда он может либо возобновить игровой процесс, либо выйти в главное меню, при этом активировав состояние завершения игрового процесса. Если во время игрового процесса персонаж игрока дошёл до финала, то игра автоматически переходит в состояние завершения игры, а если персонаж погиб, то игра переходит в состояние конца игры. В состоянии конца игры приложение ждёт от игрока нажатия на одну из двух кнопок: кнопку загрузки или кнопку выхода. При выборе первого программа перейдёт в состояние загрузки, в котором она загрузит состояние игры в точке последнего сохранения. При выборе выхода игра перейдёт в состояние завершения игры, после которого вернётся в главное меню. В состоянии завершения происходит очистка памяти компьютера от за-

груженной информации. Конечным состоянием программы является завершение работы программы. Диаграммы состояний и действий, отображающие данные процессы, представлены в приложениях А.3 и А.4.

2.1.2 Структура перцептрона

Отдельным модулем программы является модуль игрового искусственного интеллекта. Данный модуль отвечает за организацию и реализацию поведения игрового оппонента. Именно в данном модуле реализован перцептрон.

Перцептрон состоит из 4 слоёв – входного, 2 скрытых и выходного. На входном слое сети находятся нейроны, которые получают значения, соответствующие текущим характеристикам и состоянию игрового противника и объектов вокруг него, а затем преобразуют его в выходные сигналы, которые передаются на каждый нейрон следующего слоя (см. рисунок 4).

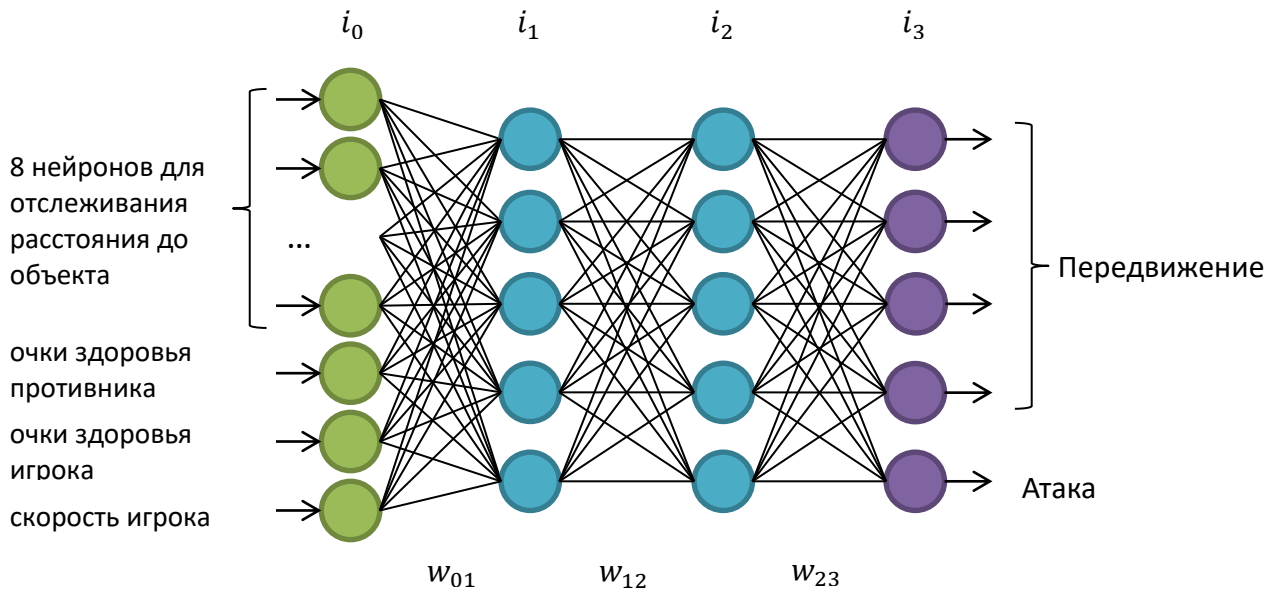

Рисунок 4 – Общая структура нейросети

Сигнал нейрона представляет собой значение функции активации f(x). В нашем случае используется следующая сигмоидальная функция:

$$f(u) = \frac{1}{1+e^{-u}}. \quad (1)$$

где u – сумма входных сигналов, e – экспонента.

Данная функция была выбрана, поскольку она обеспечивает наиболее плавную активацию, что лучше всего подходит при обработке расстояний меж-

ду управляемым объектом и другими игровыми объектами. Это позволяет задать постепенную активацию нейрона по мере приближения объекта.

Набор данных получается следующим:

– 8 рецепторов для 8 направлений, каждый из них получает значение x, которое соответствует расстоянию от управляемого объекта до другого объекта. Значение x является

– количество очков здоровья самого противника. Активация нейрона здесь происходит при уменьшении очков здоровья, поэтому формула (1) преобразуется путём разности 1-f(x), благодаря чему кривая активации «переворачивается», что позволяет обеспечить данную активацию:

$$f(x) = 1 - \frac{1}{1+e^{-(x-3)}}. \quad (2)$$

– количество очков здоровья игрока. Данный параметр необходим для системы, чтобы противник рассчитывал, когда лучше напасть на игрока, а когда – отступить. Формула активации также преобразуется и также имеет вид формулы (2).

– вектор скорости противника. Данный параметр необходим для отслеживания передвижения противника. Здесь используется формула активации (1), поскольку необходимо, чтобы нейрон активировался, если противник перемещается.

Следующим слоем являются два внутренних слоя нейронов. Именно внутри данной части перцептрона и происходят основные вычисления. Каждый нейрон текущего слоя имеет связь w с каждым нейроном предыдущего слоя. От силы этой связи зависит уровень сигнала Y, передаваемого от одного нейрона другому.

Сначала каждый нейрон первого внутреннего слоя получает сигналы от каждого нейрона входного слоя. Для каждого нейрона рассчитывается выходной сигнал. Передаточным сигналом является функция активации от суммы произведений сигнала нейрона с предыдущего слоя и веса связи между нейроном предыдущего.

$$Y_{i,j} = f(\sum_{k=1}^{n} Y_{i-1,k} * w_{kj}). \quad (3)$$

где i – номер слоя, j – номер нейрона этого слоя i, k – номер нейрона предыдущего слоя i-1, n – число нейронов в предыдущем слое.

Подобным образом второй внутренний слой считывает сигналы с первого, затем рассчитывает выходной сигнал, после чего он передаёт его на нейроны следующего выходного слоя.

Выходной слой сети состоит из нейронов, которые отвечают за поведение оппонента. Выходные данные – это команды управления. По сути – это или движение по четырем направлениям (вверх, влево, вниз, вправо), или действие атаки. Выбор действия осуществляется с помощью активации соответствующего нейрона. Активация нейрона происходит по тому же принципу, описанному ранее в формуле (3). Каждый нейрон выходного слоя принимает сигналы от всех нейронов предыдущего (второго внутреннего) слоя, умноженные на соответствующие коэффициенты синоптических связей (или на весы). Он суммирует эти сигналы, и затем происходит активация нейрона. Активация нейрона даёт понять системе, что нужно выполнить именно это действие, и она это действие и выполняет.

Принцип работы описанного выше алгоритма заключается в том, что за каждый игровой цикл перцептрон получает данные и выбирает действие, которое потом обрабатывает модуль взаимодействия игровых объектов. И то действие, которое выберет система, зависит от уровня сигналов, преобразованных из полученных данных. Эти сигналы распространяются по всей сети. На их распространение в системе влияют синоптические связи между нейронами. Изменяя данные связи, можно изменить выходные результаты. С помощью данного свойства систему можно обучить необходимому поведению.

Обучение многослойного перцептрона заключается в подстройке весов таким образом, чтобы минимизировать ошибку на обучающей выборке, и при предъявлении новых входных данных сеть вела себя как можно эффективнее и частота ошибок не выходила за границы некоего поставленного ранее предела. При этом важно соблюсти баланс, поскольку против оппонента, который прак-

тически никогда не ошибается, будет невозможно играть, а против оппонента, которого всегда выигрываешь – неинтересно. Это можно сделать, используя алгоритм обратного распространения ошибки.

Перед началом работы веса инициализируются случайным образом или берутся из файла настроек. Затем игровой оппонент, управляемый компьютером, получает состояние S0 от окружающей среды. Происходит одна итерация вычислений в сети, и в результате игровой оппонент предпринимает действие A0. Окружающая среда перемещается в новое состояние S1. В зависимости от того, что данное действие дало противнику, меняются и связи внутри нейросети. Если враг ничего не получил, то и связи между нейронами не изменяются. В случае если действие привело к нежелательному результату (например, гибели или потере очков здоровья), связи между теми нейронами, которые были активны, ослабляются. А в случае положительного эффекта (например, подбор полезных предметов, нанесение урона игроку или победа над ним) связи усиливаются.

После того, как модель обучена, необходимо понять, насколько хорошо она работает на новых данных. Для этого используется тестовая выборка данных. После тестирования необходимо оценить результаты и решить, достигнуты ли желаемые показатели. Если нет, то можно изменять различные параметры модели и пробовать обучать ее заново. Важно помнить, что для достижения желаемых результатов может потребоваться итеративный процесс изменения параметров модели и ее обучения заново.

2.2 Обзор возможностей профильного программного обеспечения

2.2.1 Программная платформа Monogame

Для работы была выбрана программная платформа (фреймворк) для разработки игр Monogame. Программная платформа Monogame представляет собой бесплатный набор инструментов с открытым исходным кодом и управляемой средой времени выполнения .NET, облегчающий разработку компьютерных игр. Данная платформа является расширенной и улучшенной версией платформы разработки Microsoft XNA, которая с 2014 года более не поддержи-

вается и не обновляется. Основные принципы данной платформы: упрощение производства кроссплатформенных приложений, предотвращение использования в играх и приложениях «повторяющегося шаблонного кода» и объединение различных аспектов разработки игр в одной системе. Данный фреймворк обеспечивает разработчиков некоторым каркасом игры на XNA, с которого можно начать процесс создания и дальнейшего развития игры. При этом данная среда позволяет создавать игры как в 2D, так и 3D пространстве.

Monogame предоставляет разработчику различные инструменты для создания контента игры. MonoGame Content Builder (MGCB) используется для помещения файлов во внутренний поток ContentPipeline. В данном потоке хранятся внешние данные, которые выводятся для работы в игре. Все данные хранятся в отдельном файле формата mgcb. В случае если необходимо отредактировать данные файлы, используется инструмент-редактор MGCB Editor. А инструмент MonoGame Effect Compiler (MGFXC) применяется при компиляции эффектов или шейдеров.

Помимо этого Monogame предоставляет и различные библиотеки с функциями и типами данных, с помощью которых можно создавать игровые объекты и описывать те или иные их свойства.

Vector2 – структура данных, которая хранит в себе значения координат местоположения объекта в окне. В структуре хранятся 2 координаты – X,Y. Каждая из координат представляет собой переменную типа float. Переменная X обозначает положение объекта по горизонтали, а Y - по вертикали. Точкой отсчёта считается левый верхний угол окна. (см. Рисунок 5). Также существует тип данных Vector3, который хранит позицию трёхмерного объекта в окне.

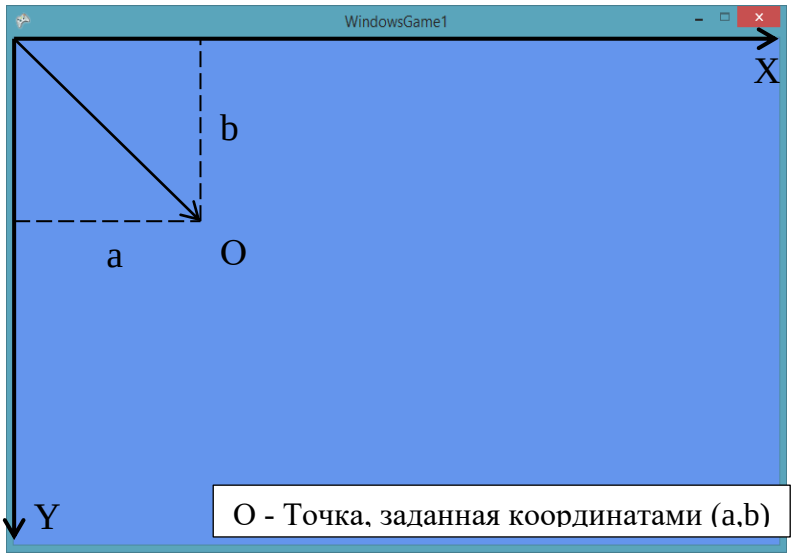

Рисунок 5 – Представление данных типа Vector2 в координатной системе

Переменная Rectangle похожа на Vector 2, но здесь координаты хранятся с помощью переменных типа int, и при этом Rectangle также хранит значения длины и ширины прямоугольника. Данный тип используется для сравнения положения одного игрового объекта относительно других. Например, если rectangle одного объекта пересекается с rectangle другого объекта, то может произойти некоторое действие (см. Рисунок 6).

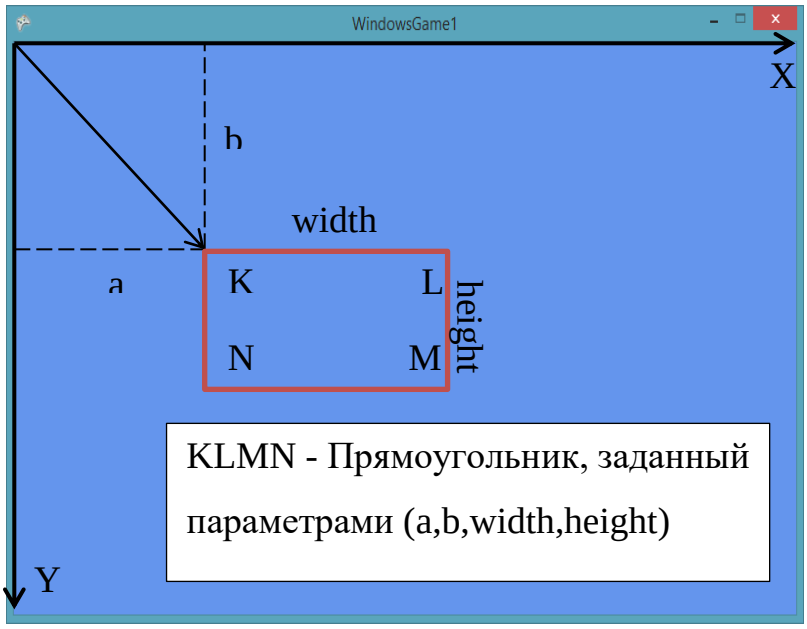

Рисунок 6 – Представление данных типа Rectangle в координатной системе

Типы KeyboardState и MouseState используются для считывания и обработки сигналов с периферийных устройств (с клавиатуры и мыши соотвественно). Переменные данного типа хранят состояние каждой кнопки периферийного устройства в данный момент, т.е. хранят массив состояний кнопок.

Также есть множество других типов, например, тип Texture2D, который используется для хранения текстур, которые необходимо вывести в окне, SoundEffect для звуковых эффектов, GameTime для хранения времени, которое прошло с момента запуска игры или с момента последнего обновления и библиотеки для эффектов DirectX. Все эти функции и типы данных можно использовать для разработки той или иной игры.

Monogame предоставляет специальные классы для разработки и работы игры. При создании нового проекта автоматически генерируются классы Game1, отвечающий за все процессы игры, и Program, который создаёт объект типа Game1 и выполняет его. Класс Game1 содержит в себе 2 переменные глобального типа, 1 конструктор и набор из пяти тесно связанных с библиотеками XNA Game Studio функций: Initialize, Load, Unload, Draw, Update.

Тип первой глобальной переменной – GraphicsDeviceManager. Она очень важна, так как обеспечивает доступ разработчику к видео устройству ПК. Объект GraphicsDeviceManager имеет свойство GraphicsDevice, которое представляет текущее графическое устройство компьютера. Так как объект графического устройства является проводником между игрой на XNA и видеокартой машины (или точнее графическим процессором (GPU) на видеокарте), все, что будет прорисовываться на экране в игре на XNA, будет проходить через этот объект. Вторая глобальная переменная – это объект класса SpriteBatch. Это основной объект, используемый для прорисовки двухмерных или трехмерных изображений (спрайтов), интегрированных в большую сцену. В двухмерных играх происходит прорисовка множества спрайтов на экране (спрайты игроков, врагов, фона и др.).

Метод Initialize используется для инициализации объектов, связанных с классом Game1. Объект графического устройства (или объект типа GraphicDevice), создаваемый в конструкторе, может быть использован в методе Initialize для инициализации объектов, которые зависят от его настроек. Метод Load загружает в память компьютера указанные в программе данные, а именно текстуры, эффекты, анимацию, звуки и т.д., для последующего использования в случае необходимости. Метод Unload напротив позволяет выгрузить из памяти ненужные файлы. Метод Draw вызывается для отображения текстур в окне приложения. Метод Update позволяет на основе полученной от пользователя информации или уже имеющейся информации обновить данные о каждом объекте и о состоянии игры в целом. Стоит отметить, что в проекте методы Load, Update и Draw присутствуют в большинстве классов, поскольку те методы, которые есть в основном классе игры, исполняются не напрямую, а через несколько перегрузок.

После первого выполнения метода Load, начинается выполнение игрового цикла. Весь игровой цикл состоит из серии методов, которые вызываются циклически, пока не закончится игра. В XNA игровой цикл состоит из методов Update и Draw. Сначала происходит обновление имеющихся данных, затем происходит прорисовка текстур по этим данным, затем вновь обновляются данные и т.д. Процесс разработки игры заключается в расписывании в Update условий, при которых будут меняться состояния объектов, и расписывании в Draw команд, которые на основе состояний объектов будут их отображать.

Для запуска игры под ОС Windows на ПК, необходима видеокарта с поддержкой драйвера WDDM 1.1 и DirectX10 или выше. Для ограниченного API (программный интерфейс приложения) подходят видеокарты с поддержкой DirectX9 и модели шейдеров 2.0.

2.2.2 Среда разработки Visual Studio

Visual Studio – программный продукт от компании Microsoft, который представляет собой интегрированную среду разработки программного обеспечения с рядом других вспомогательных инструментов. Данный продукт позво-

ляет создавать различные приложения с графическим интерфейсом для разных устройств с поддержкой технологии Windows Forms, UWP, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, фреймворками семейства .NET и т.д. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Visual Studio позволяет разрабатывать приложения и игры для всех устройств под управлением Windows. Помимо этого, она предназначена и для создания собственных или гибридных приложений для Android, iOS и Windows.

Также среда обеспечивает простую сборку, развертывание масштабируемых облачных приложений и управление ими в Azure и разработку современных веб-приложений с гибкими и многофункциональными средствами работы с открытым исходным кодом.

Visual Studio имеет и многофункциональные средства для всех типов разработки в Office. Среда позволяет не только пользоваться сторонними расширениями, но также и создавать собственные. И также среда обеспечивает про-

стую разработку и развертывание баз данных SQL Server и SQL Azure.

Среда позволяет работать с несколькими технологиями и языками. Можно использовать C++, чтобы добиться скорости, высокой производительности и совместимости на широком спектре устройств. Также возможна сборка быстрых масштабируемых кроссплатформенных приложений на JavaScript с помощью Node.js, что обеспечивает повышение продуктивности, качества и гибкости в приложениях. И ещё один поддерживаемый средой язык – Python. С его помощью можно создавать кроссплатформенные скрипты и веб-службы, и его можно использовать для создания Интернета вещей и изучения данных.

2.3 Обоснование выбора программно-технического обеспечения

Преимуществами среды Monogame являются высокая производительность, удобство разработки игр и минимальные системные требования, как для самой среды разработки, так и для создаваемых проектов, открытый иссходный код и бесплатный доступ. Однако самой главной причиной выбора данной среды является возможность написания абсолютно любого кода. Поскольку сама среда создана на базе платформы Microsoft .NET Framework, то в среде разработки Monogame используется язык C#. Данный язык сам по себе изначально задумывался как язык разработки приложений для платформы Microsoft .NET Framework, но впоследствии он был стандартизирован как отдельный самостоятельный язык. Тем не менее, одной из сфер его применения по-прежнему остаётся разработка приложений для платформы Microsoft .NET Framework. Ещё одно не менее важное свойство этого языка – его объектная ориентированность. Для создания большинства игр требуется язык, в котором реализован объектно-ориентированный подход к программированию, и язык C# как раз обладает необходимым для описания объектов и связей между ними функционалом. Данная среда идеально подходит для реализации игрового искусственного интеллекта.

Однако у среды есть и недостатки. Во-первых, немного трудная для понимания структура проекта. Во-вторых, здесь мало встроенных библиотек и поэтому здесь нет возможности создавать крупные проекты. И в-третьих, среда

привязана к платформе .NET, написанной на языке C#, поэтому она не может использовать остальные языки, например такие, как Java, C++ и т.д., что усложняет модификацию программы.

Поскольку фреймворк Monogame является улучшенной версией фреймворка XNA, который, в свою очередь, был создан на базе среды времени выполнения .NET компанией Microsoft, то и работа с ним предполагает использование среды разработки, поддерживающей .NET. Visual Studio является наиболее подходящей средой разработки, поскольку она поддерживает эту модульную платформу. При этом она также разработана компанией Microsoft, что в результате даёт наилучшую совместимость программ.

У данной системы очень много преимуществ. Эта система обеспечивает быструю навигацию, написание и исправление кода. Для этого есть большой набор инструментов, которые обеспечивают удобный поиск и навигацию в Visual Studio, и обеспечивающих переход к тому или иному типу (файлу, элементу, символу), определению, реализации, ошибке или результату в списке. Для наглядности рабочее пространство среды состоит из множества окон: обозреватель решений, окно кода, окно вывода результатов работы программы, список ошибок, Team Explorer, точки останова и т.д.

Кроме того, Visual Studio обеспечивает удобное редактирование и рефакторинг кода. Среда может предлагать быстрые действия и рекомендации по рефакторингу, показывать сведения о методе, позволяет закомментировать и раскомментировать строки, удалить или вставить текст из буфера обмена, переместить код, форматировать документ или выбранный фрагмент, сменить название переменной, инкапсулировать поле, удалить и сортировать директивы using, извлечь метод или добавить символ крышки к следующему совпадению или ко всем совпадениям и т.д.

Другим достоинством системы является простая отладка, профилирование и диагностика кода. Работоспособность кода можно проверить с помощью инструментов отладки и тестирования. Помимо отладки, запуска или остановки программы или теста для проверки можно переключать точку останова или от-

крывать окна с информацией о точках останова, выбрать шаг с обходом, заходом или выходом, задать следующий оператор и т.д. Есть возможность указать, где именно и когда необходимо прервать или приостановить выполнение приложения. Перейти в режим отладки в определенной строке гораздо проще с помощью запуска до курсора или ускоренной перемотки отладки вперед без потери точек останова с помощью принудительного запуска до курсора. При остановке кода есть и возможность проверить значения переменных для формулирования или проверки гипотезы.

При отладке в Visual Studio можно настроить уведомления, которые будут поступать при возникновении исключений, и даже выбрать специальные исключения. Получение немедленных оповещений со сведениями о внутреннем исключении и анализом пустой ссылки так же легко, как при достижении точки останова.

Есть возможность отладки многопоточного кода. Visual Studio позволяет контролировать выполнение сразу множества потоков и проверять состояние нескольких потоков, чтобы получить общее представление. Система отображает стеки вызовов всех потоков в одном графическом представлении, благодаря чему можно оценить выражения в нескольких потоках, чтобы сравнить значения, легко просматривать информацию о потоках и задачах, помечать их и замораживать. Быстро переключайтесь между контекстами выполнения и используйте маркеры потоков, чтобы в любой момент можно было проверить, какие строки кода выполняются потоками.

Предотвращение проблем с производительностью. Среда в удобном формате предоставляет информацию, которая поможет принимать верные решения по производительности при написании кода. При этом можно воспользоваться встроенной функцией редактора PerfTips и инструментами диагностики, чтобы при отладке оценить характеристики кода с точки зрения производительности и использования памяти. Также в среде есть комплексные инструменты профилирования без отладчика, чтобы лучше изучить производительность кода, включая использование ЦП, GPU и памяти, скорость отклика пользовательско-

го интерфейса и использование сети. Если необходимо проанализировать использование ЦП или отследить нежелательное выделение памяти с помощью инструмента распределения .NET, у профилировщика производительности есть инструмент для анализа производительности.

Профилировщик производительности содержит много визуализаций, таких как диаграмма пламени в средстве использования ЦП, чтобы отследить, какие пути к коду занимают большую часть времени ЦП. При обнаружении интересующих функций в стеке вызовов перейдите непосредственно к исходному коду, чтобы повысить производительность. Быстрый доступ к аналитическим сведениям на странице сводки в средстве использования ЦП. Основные функции и критические пути выделены, чтобы было удобно сразу найти причину проблем с производительностью. Есть и журнал выполнения кода .NET. В случае если при изменении кода .NET обнаруживается неожиданное поведение, ошибка, то с помощью инструментов диагностики Visual Studio и IntelliTrace можно просматривать журнал выполнения кода и переходить к проверяемому состоянию без каких-либо точек останова.

Комплексные инструменты тестирования, которые помогают писать высококачественный код. С помощью модульное тестирования можно создавать, запускать и отлаживать модульные тесты на том языке и в той тестовой среде, которые вам больше всего подходят. Обширный набор встроенных шаблонов проектов и тестовых сред подходит для множества разных платформ и позволяет начать работу без каких-либо затруднений. IntelliTest, который значительно облегчает создание и обслуживание модульных тестов для нового и существующего кода. С его помощью можно создавать входные и выходные значения для своих методов и сохраняйте их в виде небольшого набора тестов с большим объемом протестированного кода. Автоматически изменяйте их по мере совершенствования кода.

Ещё одним достоинством данной системы является её расширяемость. Во-первых, при установке системы пользователь может выбрать, какие элементы (например, библиотеки для тех или иных платформ разработки, языков про-

граммирования, технологий и т.д.) должны быть установлены. Это экономит место, занимаемое средой разработки в памяти компьютера. В случае если внезапно понадобился какой-либо элемент, можно его добавить без переустановки всей системы. Также пользователю предоставляется возможность добавить расширения, которые могут расширить функционал системы. Фреймворк Monogame и сопутствующие ему инструменты как раз можно установить таким образом. Это значительно облегчает установку и настройку фреймворка.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРЕДЛАГАЕМОГО АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ

3.1 Основные этапы практической разработки программного продукта

Предлагаемый алгоритм решения задачи разработки игрового приложения с применением многослойного перцептрона состоит из следующих шагов:

– определение параметров игры и требований к ней, в том числе параметров игрока и противника, элементов уровня, системы управления и правил игры;

– сбор и обработка данных об игре, в том числе сбор данных об игроке и противнике, оценка эффективности стратегий, выделение характеристик уровня и элементов игры;

– разработка архитектуры и модели многослойного перцептрона, включая определение числа слоев и нейронов в каждом слое, функции активации и методы обучения;

– обучение многослойного перцептрона на основе собранных данных и оптимизация его параметров для достижения максимальной эффективности при выполнении задач игрока;

– интеграция многослойного перцептрона в игровое приложение, включая совместимость с системой управления и интеграцию с другими компонентами игры;

– тестирование и оптимизация операционной скорости и надежности многослойного перцептрона на различных конфигурациях и платформах;

– оценка эффективности разработанного приложения на основе тестирования и обратной связи пользователей, а также анализ его конкурентоспособности на рынке игровых приложений.

Разработка игрового приложения с применением многослойного перцептрона требует комплексного подхода и последовательных этапов работы для

достижения максимальной эффективности и удовлетворения потребностей пользователей.

3.1.1 Программная реализация игрового приложения

Для реализации игрового приложения используется несколько классов. Основными из них являются:

– класс самой игры Game1, который по умолчанию создаётся самой платформой Monogame;

– класс уровней Level;

– класс Field для описания комнат, из которых состоит уровень;

– класс Tile для плиток, из которых состоит каждая комната;

– класс Being, который отвечает за передвигающиеся игровые объекты и включает в себя подкласс врагов Enemy и подкласс игрока Player;

– абстрактный класс всех игровых объектов GameObject, для которого классы Being и Tile являются дочерними.

Класс игры Game1 создаётся по умолчанию при создании проекта в Monogame. Класс содержит несколько переменных: объект graphics типа GraphicsDeviceManager, который хранит в себе ссылки на все находящиеся в памяти компьютера данные, объект spriteBatch типа SpriteBatch, который позволяет выводить текстуры объектов на экран. Данный класс является основным фундаментом игры, и его задачей является обеспечение работы всех внутренних методов. К внутренним методам относятся пять тесно связанных функций: Initialize, LoadContent, UnloadContent, Draw, Update. Также в данном классе существуют объекты классов Player и Level, свойства и действия которых будут описаны далее.

Метод LoadContent загружает в память компьютера указанные в программе данные, а именно текстуры, эффекты, анимацию, звуки и т.д., для последующего использования в случае необходимости. Метод UnloadContent напротив позволяет выгрузить из памяти ненужные файлы. Для загрузки файлов в методе LoadContent прописывается функция Content.Load. В ней в угловых скобках указывается класс загружаемого контента (например, <Texture2D>

для двухмерных текстур, SoundEffect для звуков, SpriteFont для определённых шрифтов и т.д.). После этого в круглых скобках указывается или только имя файла (если он находится в одной директории с приложением), или конкретный путь до файла.

Поскольку загрузка происходит для каждого игрового элемента, то и количество строк растёт при описывании функции для каждого игрового объекта. Поэтому основной задачей при написании кода является выстраивание грамотного, последовательного и компактного кода для процесса загрузки/выгрузки данных. В данном случае каждый игровой элемент является наследником класса GameObject (который будет описан далее), поэтому код загрузки для текстур записывается в общем виде для каждого игрового объекта. Поскольку все игровые объекты являются наследниками класса GameObject, они также наследуют и этот метод.

Метод Draw вызывается для отображения текстур в окне приложения. Здесь указываются, какие именно объекты будут отрисовываться и как именно. Принцип написания кода похож на метод написания кода в LoadContent/UnloadContent, и отрисовка текстур проходит с помощью методов, соответствующих отрисовываемому объекту (например, функция spriteBatch.Draw используется для отрисовывания 2D–текстур, а RectangleSprite.DrawRectangle для вывода прямоугольных фигур–примитивов). Класс GameObject также содержит наследуемый метод Draw, который является общим для всех объектов. Функция Draw принимает данные объекта (позицию), и на их основе и происходит вывод.

Метод Update отвечает за обновление состояния игровой сессии. Данная функция позволяет на основе обработанной за цикл информации обновить имеющиеся данные о каждом объекте. Именно в этом методе описывается основной принцип работы игры.

Описанные методы присутствуют в большинстве классов, поскольку те методы, которые есть в основном классе игры, исполняются не напрямую, а через несколько перегрузок. Например, метод загрузки контента LoadContent в

классе Game1 вызывает сначала метод Load в объекте класса Level. Затем внутри каждого объекта класса Level этот метод Load вызывает ещё один перегруженный метод Load, но уже для каждого объекта класса Field (комнаты), содержащегося в данном классе. Этот метод в свою очередь вызывает ещё один перегруженный метод Load для каждого объекта класса Tile (плитки), и только этот метод производит загрузку текстур плитки. По такому же принципу работают и остальные функции.

Класс Level содержит в себе информацию об уровне. Сам уровень представляет собой массив из соединённых между собой комнат. Информация о каждой комнате хранится в списке fieldList типа Field. Переменная location, находящаяся в данном классе, хранит номер комнаты, в которой находится игрок в данный момент. А переменная fieldNow типа Field – переменная, хранящая информацию о комнате, в которой находится игрок в данный момент. Данная переменная используется в алгоритме обновления информации об уровне Update. Принцип алгоритма прост: из массива выбирается комната с координатой location, и затем её данные передаются переменной fieldNow. Обновление данных происходит в этой переменной. В случае если произошёл переход из одной части уровня в другую, то новые данные, которые есть в fieldNow, передаются обратно в массив объекту комнаты под номером location. Затем значение координат меняется на новое, по этим координатам переменной fieldNow передаются данные. Это повторяется пока программа не завершит работу.

Уровень создаётся с помощью конструктора. Конструктор класса при объявлении присваивает значения всем переменным по умолчанию. Для инициализации ему необходимы два параметра – переменная типа player, которая хранит в себе состояние игрока, и строка folder, которая является наименованием директории, в которой хранятся текстовые файлы для каждой из комнат уровня. Сначала происходит инициализация списка, который будет хранить данные комнат, а затем инициализируется непосредственно каждый объект типа Field, который затем заносится в список fieldList. После этого программа по умолчанию присваивает переменной location номер 0, что означает, что игрок

начнёт игру с первой в списке комнаты.

Объекты типа Field, из которых будет состоять список, программа создаёт на основе файлов, которые хранятся в указанной директории. Программа поочерёдно открывает файлы, считывает данные, которые хранятся внутри файлов, создаёт массив символов, и передаёт их конструтору типа Field. Через этот конструктор происходит инициализация переменной Field, которая затем сохраняется в списке fieldList.

В классе содержатся методы Draw, Update и Load, которые выполняют прорисовку, обновление и загрузку данных и текстур самого уровня. А метод CheckForGoToNextLevel проверяет, дошёл ли игрок до конца уровня.

Класс Field похож на ранее описанный Level, только вместо массива комнат здесь хранится массив плиток, из которых состоит комната. Также здесь существует список находящихся в комнате врагов, предметов, дверей и передвигаемых объектов (ящиков).

В конструкторе происходит обработка массива символов, который был получены во время работы конструктора Level из файла, в котором хранится информация о размещении плиток в комнате. Каждый символ массива обозначает некоторую плитку. Например, символ «W» будет обозначать стену, символ «.» – пол; символ «M» также обозначают пол и то, что на данной плитке находится противник, а символ N обозначает выход с данного уровня и переход на следующий уровень (см. рисунок 7).

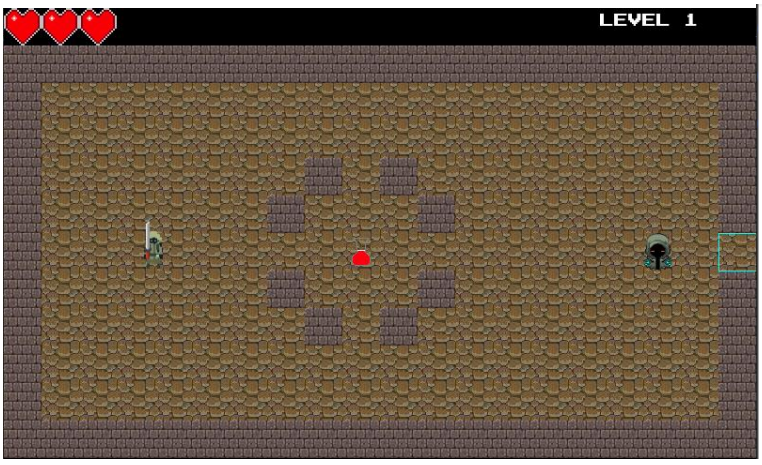

Рисунок 7 – Преобразование текстового файла в комнату

С помощью двух условий Switch происходит пошаговая обработка массива символов. Каждый символ сначала проверяется на то, к какому типу поверхности она относится, а затем проверяется на то, какой объект на этой плитке находится. Так, при символе «P» первый switch определяет, что эта плитка является полом, поэтому инициализируется плитка типа Floor, которая добавляется в список tileList, а затем второй switch определяет, что на плитке находится игровой объект типа player, который также инициализируется. Помимо это здесь также реализован скрипт, который позволяет осуществить переход между комнатами. В квадратных скобках указывается номер комнаты, в которую нужно перейти, и номер перехода из одной комнаты в другую (подразумевается, что может быть несколько переходов между двумя комнатами).

Класс Tile хранит данные о текстуре плитки, её расположении на поле, а также значения логической переменной, которая обозначает, является ли плитка проходимой или нет. Данный класс наследует свойства и методы класса GameObject и содержит только методы загрузки текстур и обновления, поскольку пока в обновлении плитки нет необходимости. Инициализация объектов данного класса происходит во время инициализации поля. Данный класс является абстрактным для разных типов классов, отвечающих за разные типы плиток, например Floor (пол) и Wall (стена).

Класс HUD хранит в себе информацию о текущем состоянии игрока, общем количестве очков здоровья, текстуры дисплея и переменные типа Vector2, отвечающие за положение элементов на экране. В конструкторе происходит присваивание переменным health и healthNow значений, отвечающих за общее и текущее значение здоровья игрока соответственно.

Класс Weapon содержит в себе переменные, которые хранят позицию оружия, его границы, очки повреждения, которые они наносят другим существам.

Методы Draw, Update и Load в обоих классах аналогично предыдущим позволяют загружать, прорисовывать и обновлять.

Класс Being отвечает за описание «живых» или передвигающихся объектов. Он наследует свойства и методы класса GameObject, а также является абстрактным для всех игровых существ, то есть для игрока и противников. У этих существ есть общие характеристики, например текстуры, позиция на экране, направление движения, переменная Rectangle, которая также как и в классе Tile, отвечает за границы объекта, общий уровень здоровья, текущий уровень здоровья, скорость, урон, оружие, которое есть у существа. Есть метод CheckBounds, который отвечает за проверку того, чтобы существо не выходило за пределы экрана. Методы avoidObstacleCollision и tileAvoiding отвечают за то, чтобы передвигающийся объект не проходил сквозь стены. Метод CheckAlive проверяет, живо ли существо.

Данный класс наследуется двумя подклассами. В классе Player хранятся основные параметры игрока. Помимо данных из Being, здесь хранятся уникальные для этого класса переменные, которых нет в классе Enemy. Первый – объект класса Hud, который отвечает за вывод информации о состоянии игрока на экран, а второй объект – объект класса keyboardState, который хранит состояние клавиатуры и позволяет управлять игроком.

В методе Update описывается модуль передвижения. Смена направления и непосредственно движение зависят от того, нажал ли пользователь кнопку, и если да, то какую. Также в методе Update описан модуль атаки. Когда игрок нажимает на определённую кнопку, то происходит атака. Если границы оружия, которые хранятся в переменной класса Weapon, соприкасаются с другим существом, то этому существу наносится урон. В методе Draw помимо отрисовки персонажа и его оружия присутствует отрисовка высвечивающегося дисплея (HUD). В методе Load происходит загрузка необходимых для игрока текстур.

3.1.2 Программная реализация перцептрона

Другой дочерний класс для класса Being – класс Enemy. Данный класс отвечает за обновление данных игрового противника. К переменным класса Being здесь добавляется параметр movable типа bool. Данный параметр указы-

вает на то, перемещается ли противник по полю или нет. В данном классе также как и в остальных методах присутствуют методы загрузки текстур Load, вывода текстур на экран Draw и метод обновления данных Update.

Остановимся подробнее на этом методе, поскольку именно в этом методе происходит основная реализация игрового искусственного интеллекта. Метод Update в классе Enemy (как и метод в классе Player) содержит модуль, который отвечает за передвижение объекта на поле. Перемещение состоит из двух элементов – обработка перцептроном игровой ситуации с последующим выбором действия и непосредственно само действие.

Как упоминалось ранее, важнейшей частью нейрона является функция активации. Её можно разбить на две части – на входную функцию, которая суммирует входные сигналы, и непосредственно функцию активации. Сначала создаётся интерфейс для входной функции, чтобы потом его можно было легко изменить в реализации нейрона. У этой функции есть только один метод – CalculateInput, который получает список подключений, описанных в интерфейсе ISynapse (Данный интерфейс представляет связи между нейронами), и затем возвращает некоторое значение на основе данных, содержащихся в списке подключений. От IInputFunction реализуется входная функция WeightedSumFunction – взвешенная суммирующая функция. Эта функция суммирует взвешенные значения по всем соединениям, которые передаются в списке.

Далее реализуется функция активации. При реализации используется тот же подход, что и при реализации функции ввода, сначала реализуется интерфейс для функций активации, а после этого могут быть сделаны конкретные реализации. Метод CalculateOutput возвращает выходное значение нейрона на основе входного значения, полученного от входной функции. В данном проекте были реализованы ступенчатая и сигмоидальная функции активации. Пороговое значение функций задается во время построения объекта, затем CalculateOutput возвращает 1, если входное значение превышает пороговое значение, иначе функция возвращает 0.

Перейдём к описанию реализации более сложных частей сети – нейронов

и соединений. Рабочий процесс, которому должен следовать нейрон, выглядит следующим образом: сначала нейрон получает входные значения от одного или нескольких взвешенных входных соединений. Эти значения передаются функции активации, которая вычисляет выходное значение нейрона. Затем полученный сигнал отправляется на выход нейрона. На основе этой абстракции рабочего процесса нейрона создается нейросеть.

Теперь подробнее рассмотрим реализацию класса нейрона. Данный класс включает в себя уникальный для каждого нейрона данные – идентификатор Id и два списка: Inputs для входных соединений и Outputs для выходных соединений. Кроме того, в данном классе есть два поля, по одному для каждой из функций, описанных ранее. Они инициализируются через конструктор. Благодаря этому можно создать нейроны с различными входными и активационными функциями.

Класс нейрона также имеет несколько методов. AddInputNeuron используется для создания входного соединения к некоторому нейрону, а AddOutputNeuron добавляет к этому нейрону выходное соединение. AddInputSynapse добавляет к нейрону InputSynapse, который является особым типом соединения. Это специальные соединения, которые используются только для входного слоя нейрона, то есть они используются только для добавления ввода во всю систему.

Последний метод, который использует данный класс: CalculateOutput. Данный метод используется для запуска алгоритма вычисления вывода. При вызове этого метода происходит вызов функции ввода, которая будет запрашивать значения для всех входящих соединений. Далее эти соединения будут запрашивать выходные значения у входных нейронов этих соединений, то есть выходные значения у нейронов предыдущего слоя. Этот процесс будет продолжаться до тех пор, пока не будет достигнут входной слой и входные значения не будут распространяться по системе.

Соединения абстрагируются через интерфейс ISynapse. Каждое соединение имеет свой вес, представленный одноименным свойством. Есть еще одна

вспомогательная функция – IsFromNeuron, которая определяет, является ли данный нейрон входным нейроном для соединения. Также реализован метод, который получает выходное значение из соединения – GetOutput.

Поля _fromNeuron и _toNeuron определяют нейроны, которые соединяет этот синапс. В дополнение к этой реализации связи есть еще связь InputSynapse, которая используется как вход в систему. Вес этих соединений всегда равен 1 и не обновляется в процессе обучения.

Далее реализуется объединение нейронов в нейронный слой. Класс содержит список нейронов, используемых в этом слое, и метод ConnectLayers, который используется для вставки двух слоев. Этот класс содержит список нейронных слоев и фабрику слоев, класс, который используется для создания новых слоев. При построении объекта в сеть добавляется начальный входной слой. Другие слои добавляются с помощью функции AddLayer, которая добавляет слой, переданный в начало текущего списка слоев. Метод GetOutput активирует выходной уровень сети, тем самым запуская цепную реакцию в сети.

Кроме того, у этого класса есть некоторые вспомогательные методы, такие, как метод PushExpectedValues, который используется для установки желаемых значений для обучающего набора, который будет передаваться во время обучения, а также метод PushInputValues, который используется для установки определенных входных данных в сеть.

Наиболее важным методом этого класса является метод Train. Данный метод отвечает за обчуение нейросети. Он получает обучающий набор и количество эпох. Для каждой эпохи он запускает весь обучающий набор по сети. Затем выходные данные сравниваются с желаемыми и вызываются функции HandleOutputLayer и HandleHiddenLayer. Эти функции реализуют алгоритм обратного распространения.

Процесс работы нейросети можно увидеть в игровом процессе. Это происходит примерно следующим образом: сначала внутри игры в конструкторе класса Game1 вместе с игроком и уровнями создается объект нейронной сети. В конструкторе выставляется, сколько во входном слое будет нейронов. После

этого добавляются слои с помощью функции AddLayer и фабрики слоев. Для каждого слоя определяется количество нейронов и функции для каждого нейрона. После завершения этой части из файла конфигураций извлекается определяются ожидаемые результаты, которыми является действие, игровой цели, и затем вызывается функция Train с входным обучающим набором и количеством эпох. Данный алгоритм как раз и отвечает за обучение.

3.2 Результаты фактического тестирования программного продукта

Подход к тестовому покрытию систем требует, чтобы все операторы в программе выполнялись хотя бы один раз, а также, чтобы выполнялись все ветви программы. При тестировании разработанной системы:

– каждый из методов класса был отдельно протестирован. Большинство методов работают исправно;

– была проведена проверка всех атрибутов, ассоциированных с объектом. В классах нет лишних атрибутов, т.е. все прописанные в классе атрибуты задействованы в работе программы;

– была проведена проверка всех возможных состояний объектов классов. Тестирование проводилось во время игрового процесса. Работа проверялась при наблюдении за состоянием объектов и его изменением в течение игрового процесса. Неисправностей обнаружено не было.

В разработанном программном продукте все классы задействованы в работе. В них нет лишних методов. Класс GameObject и абстрактный класс being имеют несколько дочерних классов (причём Being является дочерним классом GameObject). Каждый из дочерних классов работает исправно, и исправно работают их методы, унаследованные от родительских классов.

В разработанном игровом приложении есть несколько кластеров: это кластер уровней (объекты класса Level), кластер игровых полей (объекты класса Field). Игровое поле Field объединяет в кластер массивы игровых объектов (плитки, предметы, враги), и поэтому работоспособность кластера зависит от работоспособности каждого из этих элементов. Кластер Level включает в себя несколько полей, и также он обеспечивает взаимодействие уже между ними.

Поэтому его работоспособность уже зависит от исправной работы объектов Field и взаимодействия между ними.

Для объектно-ориентированных систем не подходит ни восходящая, ни нисходящая интеграция системы, поскольку здесь нет строгой иерархии объектов. Поэтому создание кластеров основывается на выделении методов и сервисов, реализуемых посредством этих кластеров. При тестировании сборки объектно-ориентированных систем используется три подхода:

– тестирование сценариев и вариантов использования. Варианты использования или сценарии описывают какой-либо один режим работы системы. Тестирование может базироваться на описании этих сценариев и кластеров объектов, реализующих данный вариант использования. В данном случае тестирование сценариев подразумевает тестирование игровых процессов и ситуаций и заключается в проверке возможных взаимодействий игровых объектов внутри одного игрового поля и внутри целого уровня. На данный момент во всех обнаруженных вариантах взаимодействия неисправностей не обнаружено. Необходимо продолжать тестирование для нахождения неопробованных вариантов;

– тестирование потоков. Этот подход основывается на проверке системных откликов на ввод данных или группу входных событий. Объектно-ориентированные системы, как правило, событийно-управляемые, поэтому для них особенно подходит данный вид тестирования. При использовании этого подхода необходимо знать, как в системе проходит обработка потоков событий. В данном случае это обработка файлов и ввода с клавиатуры игрока. На данный момент все системы работают исправно;

– тестирование системы заключается в полной проверке работоспособности приложения как единой системы. Как правило, тестирование совершается методом «чёрного ящика», то есть с помощью проверки работоспособности без исходного кода, или глазами обычного пользователя. Результаты тестирования совмещаются с результатами тестирования вышеописанных элементов, и на их основе делается окончательный вердикт.

На данный момент система работает стабильно. Критических неисправностей обнаружено не было. Есть неисправности в методах некоторых классов, но на данный момент они не влияют на ход работы программы. С внесением небольших исправлений программа будет полностью готова к использованию.

3.3 Анализ достоверности и практической значимости результатов

Созданный проект может найти применение не только в сфере медиа развлечений, но и в других сферах. Так, моделирование поведения игровых объектов можно преобразовать в модель поведения виртуальных объектов, моделирующих некие реальные процессы.

Модель поведения, созданная с помощью нейросети, вполне возможно применить в различных виртуальных тренажёрах. С её помощью можно смоделировать поведение объектов или протекание процессов, которые будут максимально приближены к реальным. Это также поможет разнообразить тренировки, поскольку в данном случае поведение объектов станет менее предсказуемым, что сделает тренировки более разнообразными, а ситуации менее линейными.

С помощью созданной модели поведения возможно ние заболеваний, моделирование процесса их распространения (причём как в одном организме, так и в группе). Благодаря этому можно лучше спрогнозировать поведение болезни, и благодаря этому составить наиболее оптимальный курс лечения.

Прогнозирование погоды также возможно с помощью нейросетевой модели поведения. В данном случае будет рассматриваться и прогнозироваться протекание процессов в атмосфере. Однако на данный момент уже существуют некоторые нейросети, которые этим успешно занимаются. Этим занимается, например, «Яндекс.Погода»: там используется специальный алгоритм на основе нейронных сетей, который прогнозирует метеорологические изменения с точностью до минут (или почти так).

Разработанную модель можно использовать и как инструмент для определения интенсивности дорожного трафика, и как непосредственно участника

этого движения. Внедрение нейросетей в транспортные средства в качестве автопилота, который постоянно анализирует получаемую информацию и выбирает оптимальный маршрут движения, поможет улучшить и обезопасить дорожное движение.

Использование нейросетей возможно и в качестве одного из элементов робототехники. Модель поведения можно использовать для управления роботами (например, прокладка маршрута) и манипуляторами.

Политологические и социологические исследования: здесь нейросеть может применяться для предсказания общественных и политических событий, а также их изучения и визуализации.

Безопасность и охрана: здесь возможно прогнозирование распространения природных и стихийных бедствий.

Поведение животных также можно смоделировать с помощью нейросетей. Поскольку строение животного организма схоже с человеческим, то и процессы, протекающие в головном мозге человека во время принятия решений и совершения действий, схожи с животными. Отличие заключается в комплексности процессов. Благодаря нейросетям, можно не только спрогнозировать поведение одного животного, но и небольших групп или целых ареалов.

Таким образом, моделирование поведения с помощью нейросетей может найти применение во многих областях, поскольку технология нейросетей сама по себе довольно гибка, и применяема в любой сфере.

ЗАКЛЮЧЕНИЕ

В данной работе были изучены возможности применения ИИ в игровых приложениях, а также был проведён анализ существующих подходов к использованию нейронных сетей в игровых приложениях в качестве системы игрового искусственного интеллекта. Выявлено, что на данный момент существуют различные реализации подобных систем и в большинстве случаев в них используются перцептроны.

Далее был определён функционал игрового приложения, на основании которого были сформулированы требования к программному продукту, соблюдение которых позволило создать полноценный и работоспособный продукт.

Затем было выполнено проектирование программного продукта модульной архитектуры, продумано взаимодействие модулей.

Спроектирована структура нейронной сети, продумано её применение в модели поведения игрового персонажа. Было выбрано два внутренних слоя и с пятью количеством нейронов в каждом из них, а также для всех нейронов были выбраны типы входных данных и функции активации нейронов. Затем была проведена реализация нейронной сети с помощью языка C#.

В итоге приложение состоит из шести модулей, одним из которых является модуль игрового искусственного интеллекта. На вход системы подаются файлы конфигурации и уровней, а также пользовательский ввод, на выходе система выдаёт изображение игрового процесса.

Также была выполнена разработка, отладка и тестирование игрового приложения. Проведена проверка работоспособности как самого игрового приложения, так и созданного перцептрона.

На данный момент система работает с некоторыми ошибками. В будущем возможно улучшение данного приложения путем расширения функциональности, исправления ошибок и добавления новых возможностей для персонализации игрового процесса, а также улучшения точности определения поведения игрока.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1 Аверкин А.Н., Гаазе-Рапопорт М.Г., Поспелов Д.А. Толковый словарь по искусственному интеллекту М.:Радио и связь, 1992. 256 с.

2 Wayback Machine [Электронный ресурс] Jack Copeland What is Artificial Intelligence? Режим доступа: https://web.archive.org/web/20081224114209/http://www.alanturing.net/turing_archive/pages/Reference%20Articles/what_is_AI/What%20is%20AI09.html

3 Нейронная сеть // Большая российская энциклопедия : [в 35 т.] / гл. ред. Ю. С. Осипов. – М. : Большая российская энциклопедия, 2004 – 2017.

4 Аверкин А.Н., Гаазе-Рапопорт М.Г., Поспелов Д.А. Толковый словарь по искусственному интеллекту М.:Радио и связь, 1992. 256 с.

5 Mitchell, Tom (1997). Machine Learning. New York: McGraw Hill. ISBN 0-07-042807-7. OCLC 36417892. Archived from the original on 2020-04-07. Retrieved 2020-04-09.

6 Stopgame.ru [Электронный ресурс] Нейронные сети в игровой индустрии. Режим доступа: https://stopgame.ru/blogs/topic/107976/neyronnye_seti_v_igrovoy_industrii

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Аверкин, А.Н. Толковый словарь по искусственному интеллекту / А.Н. Аверкин, М.Г. Гаазе-Рапопорт, Д.А. Поспелов – М.:Радио и связь, 1992. – 256 с.

2 Берг, Д.Б. Модели жизненного цикла : учебное пособие / Д.Б. Берг, Е.А. Ульянова, П.В. Добряк – Екатеринбург : Изд-во Урал. ун-та, 2014. – 74 с.

3 Брауде, Э. Технология разработки программного обеспечения : пер. с англ. / Э. Брауде – СПб. : ПИТЕР, 2004. – 655 с.

4 Бутл, Р. Искусственный интеллект и экономика. Работа, богатство и благополучие в эпоху мыслящих машин / Р. Бутл – М.: Интеллектуальная Литература, 2022. – 432 с.

5 Буч, Г. Язык UML. Руководство пользователя : Пер. с англ. Мухин Н. / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд. – М.: ДМК Пресс, 2006. –496 с.

6 Буч, Г. UML. Классика CS / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд. – СПб.: «Питер», 2006. – 736 с.

7 Васильев, А.Н. Программирование на C# для начинающих. Основные сведения / А.Н. Васильев – М. : Бомбора, 2022. – 592 с.

8 Васильев, Ф.П. Методы оптимизации. / Ф.П. Васильев – М. : Факториал Пресс, 2002. – 824 с.

9 Вигерс, К. Разработка требований к программному обеспечению пер. с англ. / К. Вигерс – М. : «Русская Редакция», 2004. – 576 с.

10 Гаврилова, Т.А. Базы знаний интеллектуальных систем: Учебник для вузов / Т.А. Гаврилова, В.Ф. Хорошевский – СПб.: Питер, 2000 – 384 с.

11 Гайдуков, С.А. Введение в XNA / С.А. Гайдуков. – М.: Интернет-Университет Информационных Технологий, 2009. – 563 с.

12 Галушкин, А. И. Нейронные сети: основы теории / А. И. Галушкин – М. : Горячая линия – Телеком, 2012. –496 с.

13 Горнаков, С.Г. Программирование компьютерных игр под Windows в XNA Game Studio Express. / С.Г. Горнаков – М.: ДМК Пресс, 2008. – 384 с.

14 Документация Monogame [Электронный ресурс] Режим доступа: https://docs.monogame.net/ – 10.03.2023

15 Жданов, А.А. Автономный искусственный интеллект / А.А. Жданов – М.: БИНОМ. Лаборатория знаний, 2009. – 359 с.

16 Зиборов, В. Visual C# 2010 на примерах / В. Зиборов – СПб. : БХВ-Петербург, 2011 – 470 с.

17 Каллан, Р. Основные концепции нейронных сетей / Р. Каллан. - М : Вильяме, 2001. -288 с.

18 Кащеев, П.О. Применение технологий искусственного интеллекта при моделировании поведения виртуальных объектов. / П.О. Кащеев // Материалы XVI международной научной конференции «Системный анализ в медицине» (САМ 2022) / под общ. ред. В.П. Колосова. – Благовещенск: ДНЦ ФПД, 2022 – С. 89-91.

19 Кащеев, П.О. Особенности разработки игрового приложения с применением технологий искусственного интеллекта / П.О. Кащеев // Инновационный потенциал развития науки в современном мире: технологии, инновации, достижения / Сборник научных статей по материалам XII Международной научно-практической конференции (18 апреля 2023 г., г. Уфа) – Уфа: Изд. НИЦ Вестник науки, 2023. – С. 85-88.

20 Кормен, Т. Алгоритмы: построение и анализ = Introduction to Algorithms / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; под ред. И. В. Красикова. – 2-е изд. – М. : Вильяме, 2005. – 1296 с

21 Курейчик, В.М. Эволюционная адаптация в обучении нейронных сетей / В.М. Курейчик, Б.К. Лебедев, В.И. Божич // Известия Южного федерального университета ; технические науки. - Таганрог : Изд-во ТТИ ЮФУ, 1999.

22 Левитин, А.В. Алгоритмы : введение в разработку и анализ / А.В. Левитин. – М. : Вильяме, 2006. – 576 с.

23 Леоненков, А. Самоучитель UML / А. Леоненков – 2-е изд., перераб и доп. – СПб. : БХВ-Петербург, 2004. – 432 с.

24 Липаев, В.В. Надежность программных средств / В.В. Липаев. – М.: СИНТЕГ, 1998. – 358 с.

25 Макконнелл, С. Совершенный код. Мастер-класс : пер. с англ. / С. Макконнелл – М. : «Русская редакция», 2010. – 896 стр.

26 Макки, А. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов / А. Макки – М. : Apress, 2010. – 416 с.

27 Михайлов, М.Н. Имитационное моделирование / М.Н. Михайлов, Т.В. Первозванская, Л.П. Вьюненко. – СПб: Юрайт, 2017. – 284 с.

28 Нейронная сеть // Большая российская энциклопедия : [в 35 т.] / гл. ред. Ю.С. Осипов. – М. : Большая российская энциклопедия, 2004 – 2017.

29 Нильсон, Н. Искусственный интеллект. – М.: Мир, 1973. – 273 с.

30 Орлов, С.А. Технологии разработки программного обеспечения / С.А. Орлов. – СПб. : ПИТЕР, 2002. – 464 с.

31 Осовский, С. Нейронные сети для обработки информации / С. Осовский. – М. : Финансы и статистика, 2004. – 344 с.

32 Официальный сайт Visual Studio [Электронный ресурс]: – Режим доступа: https://visualstudio.microsoft.com/ru/vs/features/?tab=features – 25.02.2023

33 Рассел, С. Искусственный интеллект: современный подход: пер. с англ. и ред. К. А. Птицына. / С. Рассел, П. Норвиг – 2-е изд. – М.: Вильямс, 2006. – 1408 с.

34 Рид, А. Изучаем XNA 4.0, O'Reilly Media: 2010, 430с

35 Редько, В. Г. Эволюция, нейронные сети, интеллект : модели и концепции эволюционной кибернетики / В. Г. Редько. – М. : УРСС, 2005. – 224 с.

36 Рудой, Г.И. Выбор функции активации при прогнозировании нейронными сетями / Г. И. Рудой // Машинное обучение и анализ данных. - 2011. - Т. 1. № 1. - С. 16-39.

37 Рутковский, Л. Методы и технологии искусственного интеллекта / Л. Рутковский. – М. : Горячая линия-Телеком, 2010. – 520 с.

38 Смолин, Д.В. Введение в искусственный интеллект: конспект лекций / Д.В. Смолин – М.: ФИЗМАТЛИТ. – 208 с.

39 Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core : пер. с англ. / Э. Троелсен, Ф. Джепикс. – 8-е изд. – СПб. : ООО “Диалектика”, 2018. – 1328 с.

40 Фихтенгольц, Г. М. Курс дифференциального и интегрального исчисления. В 3 т. Т. 3 / Г. М. Фихтенгольц. – М. : Физматлит, 2008. – 728 с.

41 Флах, П. Машинное обучение. – М.: ДМК Пресс, 2015. – 400 с.

42 Хайкин, С. Нейронные сети: полный курс, 2-е изд., испр.: Пер. с англ./ С. Хайкин – Рязань, 2003. – 140 с.

43 Alippi, C. Simple approximation of sigmoidal functions : realistic design of digital neural networks capable of learning / C. Alippi, G. Storti-Gajani // Proceedings of the IEEE International symposium on curcuits and systems (Singapore, 1991). 1991 – 1505-1508 с.

44 Blanchard, B.S. Systems engineering and analysis / B.S. Blanchard, Wolter J. Fabrycky. – 4-е изд. – Хобокен : Prentice Hall, 2006 – 824 с.

45 Bishop, C.M. Neural networks for pattern recognition / C. M. Bishop. – Oxford : University press, 1995. – 482 с.

46 Cybenko, G. Approximation by superposition of a sigmoidal functions / G. Cybenko // Mathematics of control, signals, and systems. – 1989. – 303-314 с.

47 OMG Unified Modeling Language (OMG UML), Infrastructure Version 2.2 [Электронный ресурс] : офиц. сайт. – 2009 – Режим доступа: https://www.omg.org/spec/UML/2.2/Superstructure/PDF – 20.02.2023

48 Selby, R.W. Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research / R.W. Selby – Нью-Йорк: John Wiley & Sons, 2007. – 834 с.

49 Wayback Machine [Электронный ресурс] John McCarty What is Artificial Intelligence? Режим доступа: https://web.archive.org/web/20151118212402/http://www-formal.stanford.edu/jmc/whatisai/whatisai.html

50 Wayback Machine [Электронный ресурс] Jack Copeland What is Artificial Intelligence? Режим доступа: https://web.archive.org/web/20081224114209/http://www.alanturing.net/turing_archive/pages/Reference%20Articles/what_is_AI/What%20is%20AI09.html

51 Wikipedia [Электронный ресурс]: Microsoft XNA – Режим доступа: https://en.wikipedia.org/wiki/Microsoft_XNA – 27.02.2023

52 Winston, R. Managing the Development of Large Software Systems [Электронный ресурс] Режим доступа: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/ – 28.02.2023

ПРИЛОЖЕНИЕ А

Диаграммы проиложения

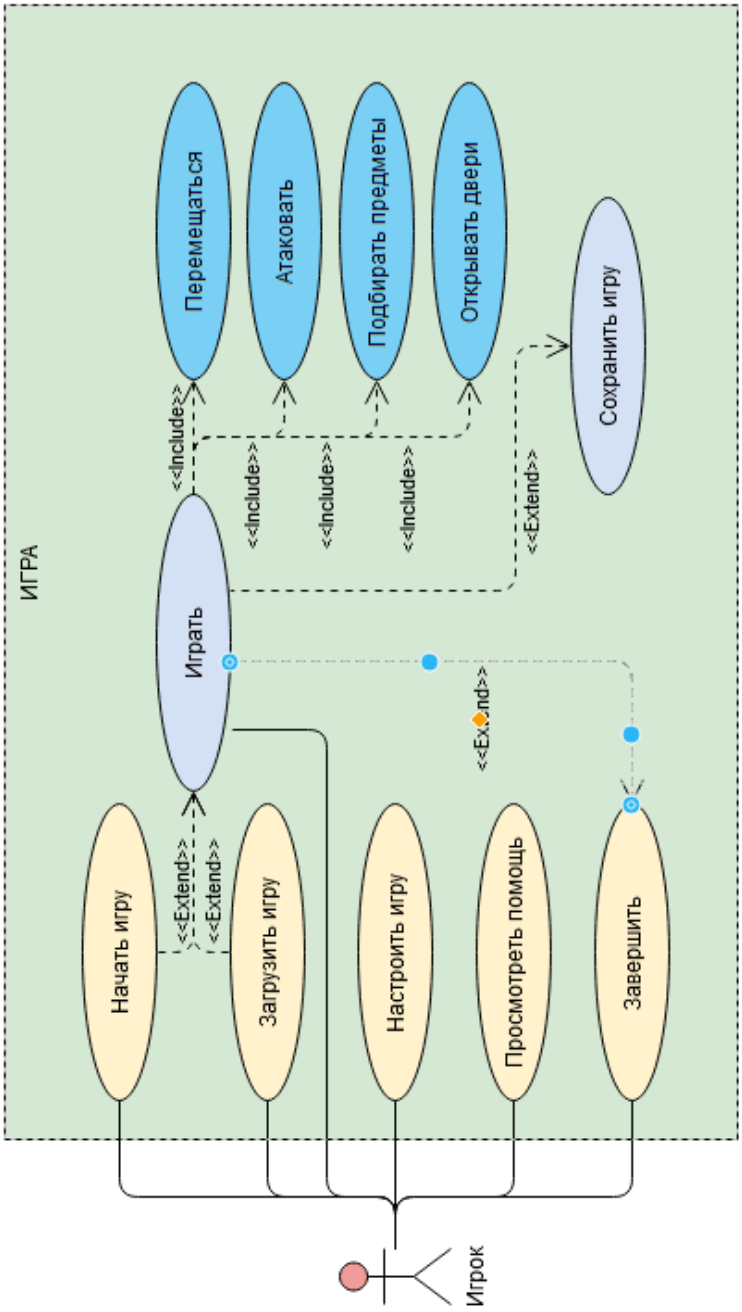

Рисунок А.1 – Диаграмма вариантов использования

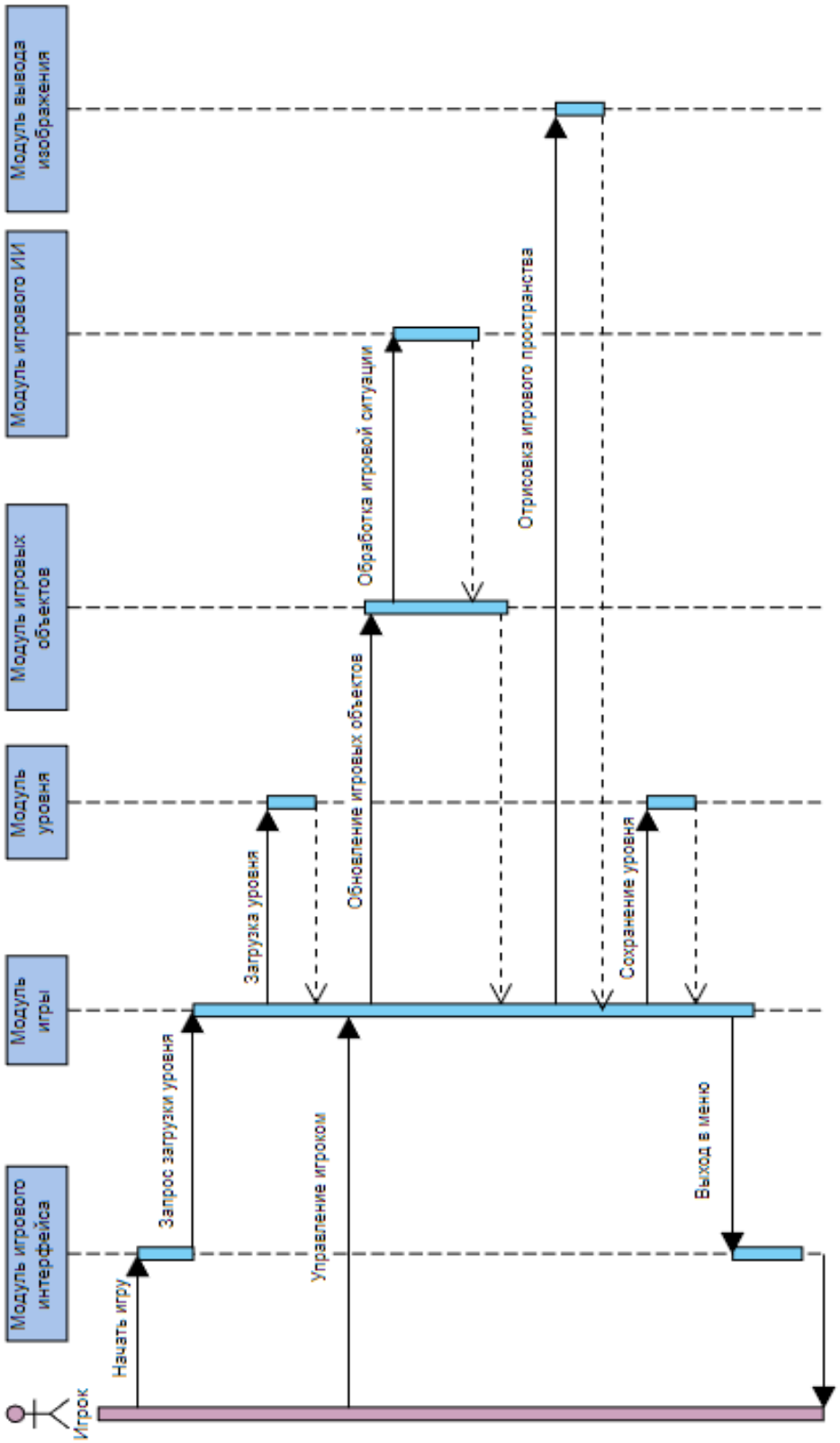

Рисунок А.2 – Диаграмма последовательности

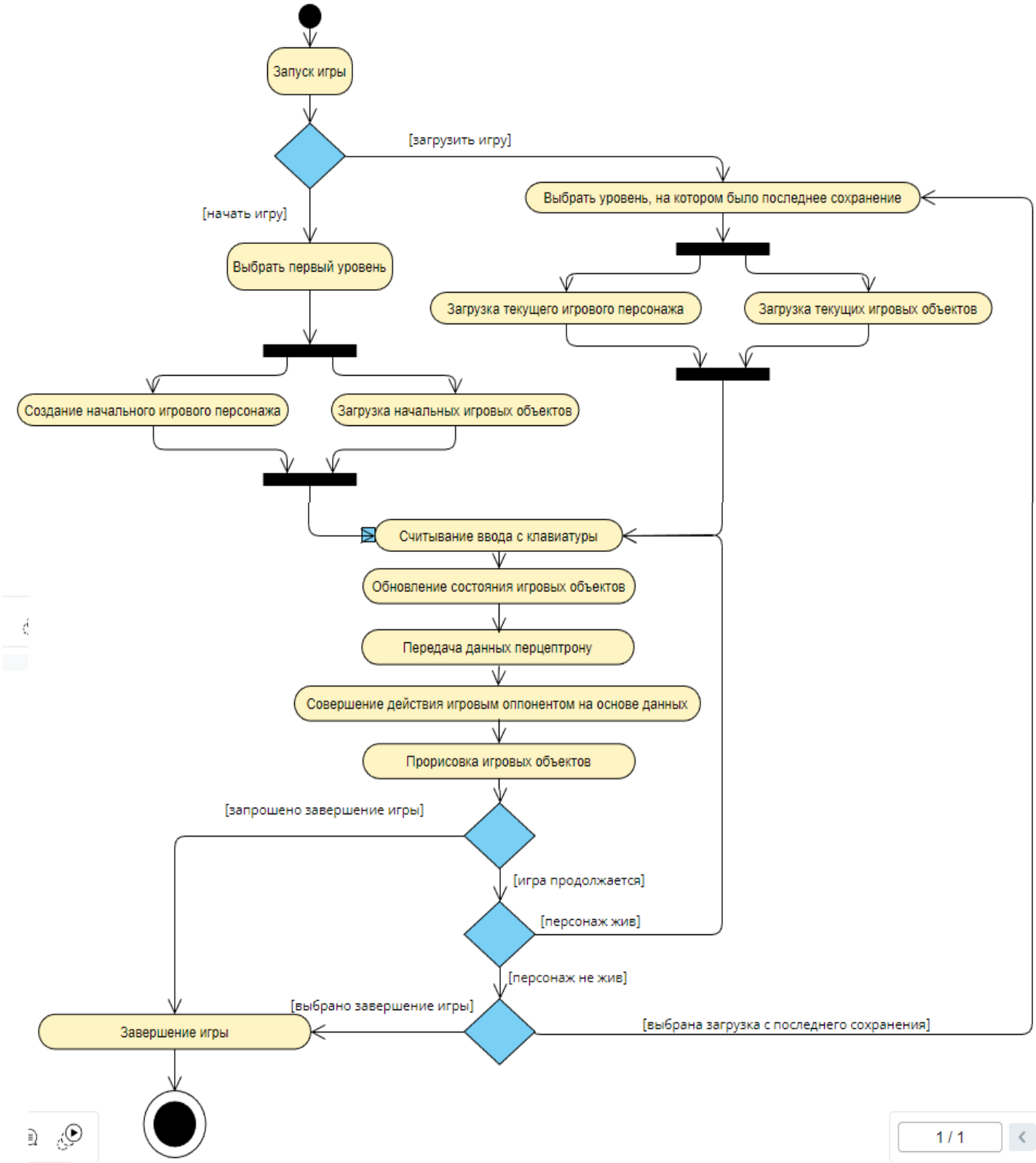

Рисунок А.3 – Диаграмма действий

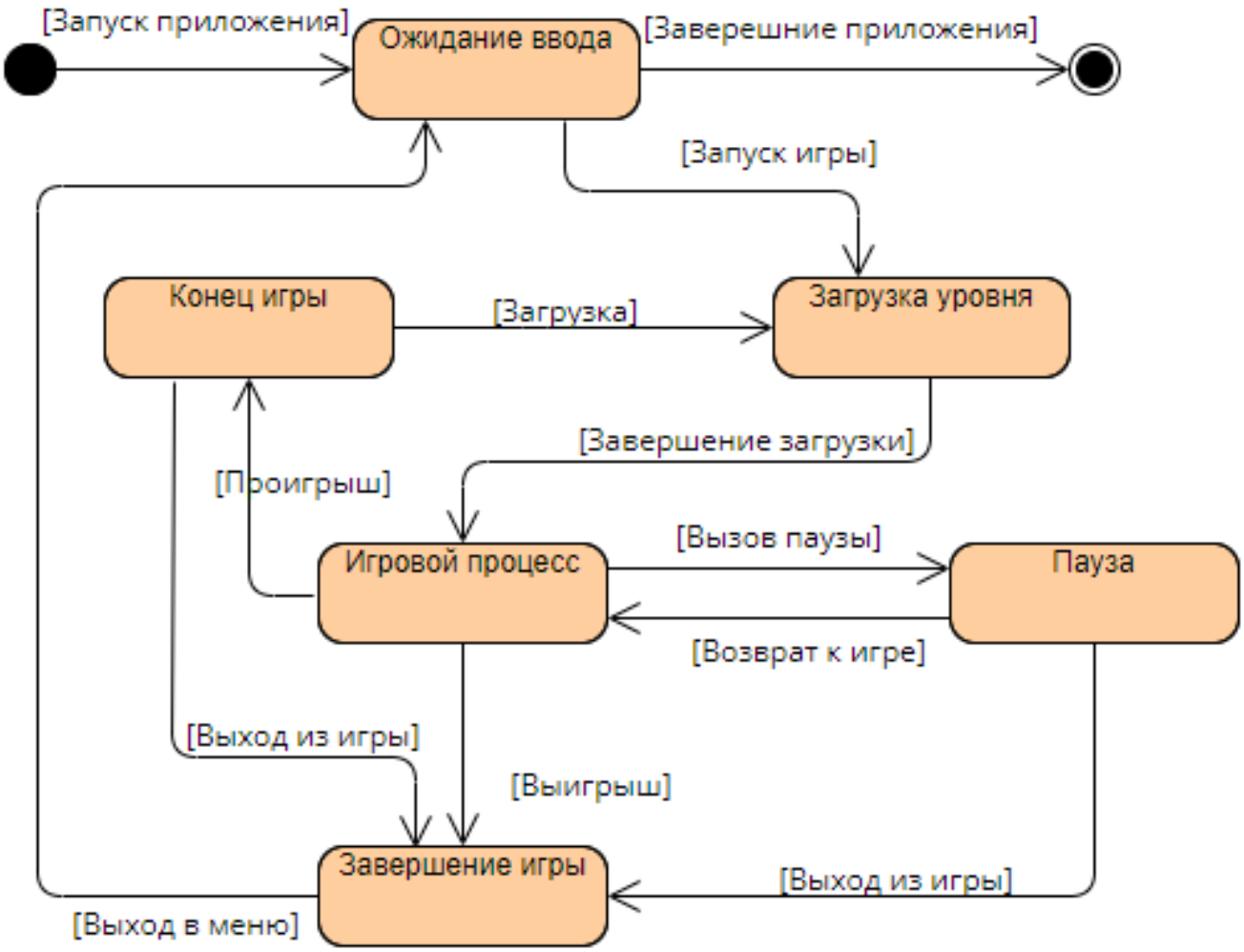

Рисунок А.4 – Диаграмма состояний