МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет (национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д. В. Топольский
«___»_____ 2022 г.

ПРОЕКТИРОВАНИЕ МОДЕЛИ ПОВЕДЕНИЯ МОБОВ В КОМПЬЮТЕРНЫХ ИГРАХ, РАЗРАБОТАННЫХ НА БАЗЕ UNITY, С ПОМОЩЬЮ НЕЙРОСЕТИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-09.03.01.2022.308 ПЗ ВКР

Руководитель работы,
к.п.н., доцент каф. ЭВМ
_____Ю. Г. Плаксина
«___»_____2022 г.

Автор работы,
студент группы КЭ-406
_____И.Ю. Сорокин
«___»_____2022 г.

Нормоконтролёр,
к.п.н., доцент каф. ЭВМ
_____ М. А. Алтухова
«___»_____2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Сорокину Илье Юрьевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. Тема работы: «Проектирование модели поведения мобов в компьютерных играх, разработанных на базе Unity, с помощью нейросети» утверждена приказом ректора Южно-Уральского государственного университета от 12 декабря 2021 г. №308/141

2. Срок сдачи студентом законченной работы: «2» июня 2022 г.

3. Исходные данные к работе:

Разрабатываемая система должна обеспечивать следующие возможности:

–нейросеть должна создавать новые модели поведения;

–в системе должно иметься хранилище данных, в которой хранятся две модели поведения, первоначальная и новая;

–система должна оценивать эффективность модели поведения;

–коэффициенты эффективности зависят от результатов тестовых боев уничтоженных противников, членов собственной команды и их выживания;

–система должна сравнивать новые моделями поведения со старыми по значению коэффициента эффективности;

–система должна удалять устаревшие модели поведения;

–каждая модель поведения должна обеспечивать поиск противника, его уничтожение, а также предотвращение собственного уничтожения;

–должно иметься не менее двух команд;

–команды должны быть враждебны друг к другу;

–в процессе обучения нейросети в симуляции должно иметься не менее двенадцати NPC, разделенных на две команды;

–команды должны иметь равное количество NPC;

–одна из команд будет получать новые модели поведения из нейросети, в то время как другая будет сохранять изначальную модель поведения;

–время симуляции не должно превышать 5 минут;

–система должна иметь возможность ускорения симуляции;

–система должна иметь возможность остановки симуляции;

–пользователь должен иметь возможность наблюдения за симуляцией;

–система должна осуществлять запись коэффициентов эффективности в отчет;

–разработка быть работоспособной на windows 7, 8, 8.1, 10;

–разработка должна вестись на языке c#.

4. Перечень подлежащих разработке вопросов:

–рассмотрение существующих программных решений, реализующих нейросеть для моделирования поведения мобов;

–разработка структуры нейросети способной моделировать поведение мобов;

–обучение нейросети;

–оценка работоспособности разработанной модели поведения.

5. Дата выдачи задания: «1» декабря 2022 г.

Руководитель работы /*Ю.Г. Плаксина*/

Студент /*И.Ю. Сорокин* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение	12.04.2022	
Обзор литературы. Рассмотрение существующих программных решений, реализующих нейросеть для моделирования поведения мобов	23.04.2022	
Разработка структуры нейросети способной моделировать поведение мобов	10.05.2022	
Обучение нейросети	15.05.2022	
Оценка работоспособности разработанной модели поведения	22.05.2022	
Компоновка текста работы и сдача на нормоконтроль	23.05.2022	
Подготовка презентации и доклада	31.05.2022	

Руководитель работы _____/ *Ю.Г. Плаксина* /

Студент _____/ *И.Ю. Сорокин* /

Аннотация

И. Ю. Сорокин Проектирование модели поведения мобов в компьютерных играх, разработанных на базе Unity, с помощью нейросети. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2022, 46 с., 6 ил., библиогр. список – 17 наим.

В рамках выпускной квалификационной работы был проведён анализ существующих аналогов системы. В результате анализа были выявлены недостатки этих систем. Определены пути и решения устранения выявленных недостатков в разрабатываемой системе. Рассмотрены основные технологии, применяющиеся в разработке схожих систем, и были выбраны наиболее подходящие для данного проекта.

После анализа были произведены проектирование, разработка и тестирование нейросети для моделирования поведения мобов в компьютерной игре.

Результатом работы являются нейросеть, реализующая заявленный функционал.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сфера развлечений — одна из важнейших в повседневной жизни человека, существенно влияющая на состояние общества. Ежегодно доход данной отрасли составляет порядка 2 триллионов долларов.

Компьютерные игра – компьютерная программа, служащая для организации игрового процесса, геймплея, связи с другими пользователями, игроками, или с самой собой.

Основная цель компьютерных игр – организация досуга игрока, посредством решения некоторых логических задач, проверки реакции, внимания или находчивости.

За 50 лет со дня своего основания, индустрия компьютерных игр стала неотъемлемой частью жизни общества, затронувшей множество людей. Даже не интересующиеся данной областью люди имеют некоторое представление о компьютерных играх.

Модернизация процесса разработки является одним из основных направлений развития компьютерных игр. Улучшение данного аспекта позволит сократить требуемое количество времени и ресурсов, выделяемых для этой задачи.

Существует большое количество способов создания компьютерных игр. Для упрощения и оптимизации процесса, а также улучшения качества, было разработано различное программное обеспечение и вспомогательные пакеты. Одним из таких решений является платформа Unity, рассматриваемая в данной работе.

Преимуществами Unity являются простота использования, обширная база различных библиотек, ускоряющих и упрощающих процесс разработки, встроенная система управления физикой объектов, мультиплатформенность, частые обновления, оптимизирующие работу и исправляющие найденные ошибки.

Существенную часть времени разработки компьютерных игр составляет проектирование модели поведения мобов.

Использование нейросети позволит автоматизировать процесс разработки моделей поведения мобов, что позволит ускорить создание компьютерных игр.

Основной проблемой применения нейросети для моделирования поведения мобов является ресурсная и аппаратная затратность. Непрерывный процесс улучшения моделей поведения для мобов нейросетью приводит к необоснованно высокой сложности игры. Так в 2018 году нейросеть одержала серию побед в игре Starcraft 2, ни разу не проиграв сборной профессиональных киберспортсменов.

В данной работе планируется уменьшить количество необходимых входных данных, посредством разработки нейросети на базе first-person-shooter (FPS), что позволит получить необходимые данные за короткий промежуток времени из большого количества смертей мобов. Сложность поведения будет регулироваться заранее установленным пределом.

Цель работы: Создание модели поведения мобов посредством нейросети.

Задачи работы:

– провести обзор аналогичных решений и осуществить постановку задачи;

– провести обзор современных средств реализации;

– провести анализ требований и спроектировать нейросеть;

– обучить нейросеть;

– провести тестирование полученных, с помощью нейросети, моделей поведения.

Работа состоит из 6 глав. Первая глава содержит анализ предметной области и выборе основных технологических решений. Во второй главе выделены требованиях, поставленных перед проектом. В третьей главе представлен алгоритм решения задачи. В четвертой главе описана реализация

поставленной задачи. В пятой − главе проведена процедура тестирования разработанного продукта. Шестая глава − вывод по работе.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Существует несколько примеров использования нейросети для игры против человека, например AlphaStar [1], AlphaGo [2] и OpenAI Five [3].

При разработке AlphaStar на вход подавалось большое количество примеров игр киберспортсменов, на основе которых нейросеть составляла тактику игры. После этого нейросеть проводила тестовые бои сама с собой, в ходе чего разрабатывались новые тактики игры, а старые замораживались. Далее процесс повторялся.

Недостатками данного решения являются чрезмерно большой объем входных данных, значительный объем данных, накапливающийся в базе данных, и высокий уровень сложности полученной модели поведения, что сделала победу невозможной даже для профессиональных киберспортсменов.

Аналогичным образом обучалась и AlphaGo. Данная нейросеть была разработана компанией DeepMind для игры в го. Однако, из-за чрезмерно большого количества возможных комбинаций, разработчики решили, что будет целесообразно подать на вход только самые часто используемые, что уменьшило количество входных данных.

AlphaGo имела те же недостатки, что и AlphaStar, а именно большой объем требуемых входных данных и высокая сложность игры. Так в 2016 году AlphaGo победила действующего мирового чемпиона Ли Седоля [4].

Иной метод обучения был у OpenAI Five, разработанной компанией OpenAI для игры в Dota 2. В данную нейросеть не было заложено каких-либо моделей поведения или примеров тактик игроков. OpenAI Five самостоятельно сформировала модель поведения с нуля, в процессе проведения боев против ботов.

В отличии от аналогов, в OpenAI Five проблема большого объема входных данных была решена, однако это привело к увеличению времени обучения.

Проблема высокой сложности не была решена, в 2019 году OpenAI Five обыграла действующих чемпионов со счетом 2:0 [5].

Таким образом, можно сформулировать критерии, для оценки упомянутых аналогов. Данные критерии представлены ниже (таблица 1).

Таблица 1 – Сравнение аналогов

Нейросеть	**Обучение**	**Ограничение на сложность игры**
AlphaStar	Формирование модели поведения из полученных примеров игр и бои самой с себя	Отсутствует
AlphaGo	Формирование модели поведения из полученных тактик игры и бои самой с себя	Отсутствует
OpenAI Five	Бои против ботов и самой себя	Отсутствует

Разрабатываемая модель сравнивалась с AlphaStar так как она считается наиболее известной и продвинутой нейросетью, разработанной для данной цели.

1.2 Анализ основных технологических решений

В большинстве игр поведение мобов задается скриптами. Однако, есть несколько примеров использования нейросети, от шахмат и го, до Doom, Starcraft, Dota.

Скрипт представляет из себя сценарий, некоторый набор правил и логику, как игровой объект будет реагировать на ту или иную ситуацию.

Плюсами мобов, управляемых скриптами, является простота разработки и внесения изменений. Их недостатками является отсутствие самостоятельного развития и невозможность реагирования на не прописанную заранее ситуацию.

В свою очередь нейросеть способна самостоятельно развиваться, находить более эффективный путь к решению задачи и возможность обработки ситуаций реакции на, которые не были заранее подготовлены. Минусами

нейросетей является их сложность разработки, возможная высокая затратность системных ресурсов, а также требуется время на их обучение.

Под искусственной нейронной сетью понимают математическую модель, которая построена и работает по принципу функционирования биологических нейросетей.

Общая модель нейрона представлена ниже, на рисунке 1.

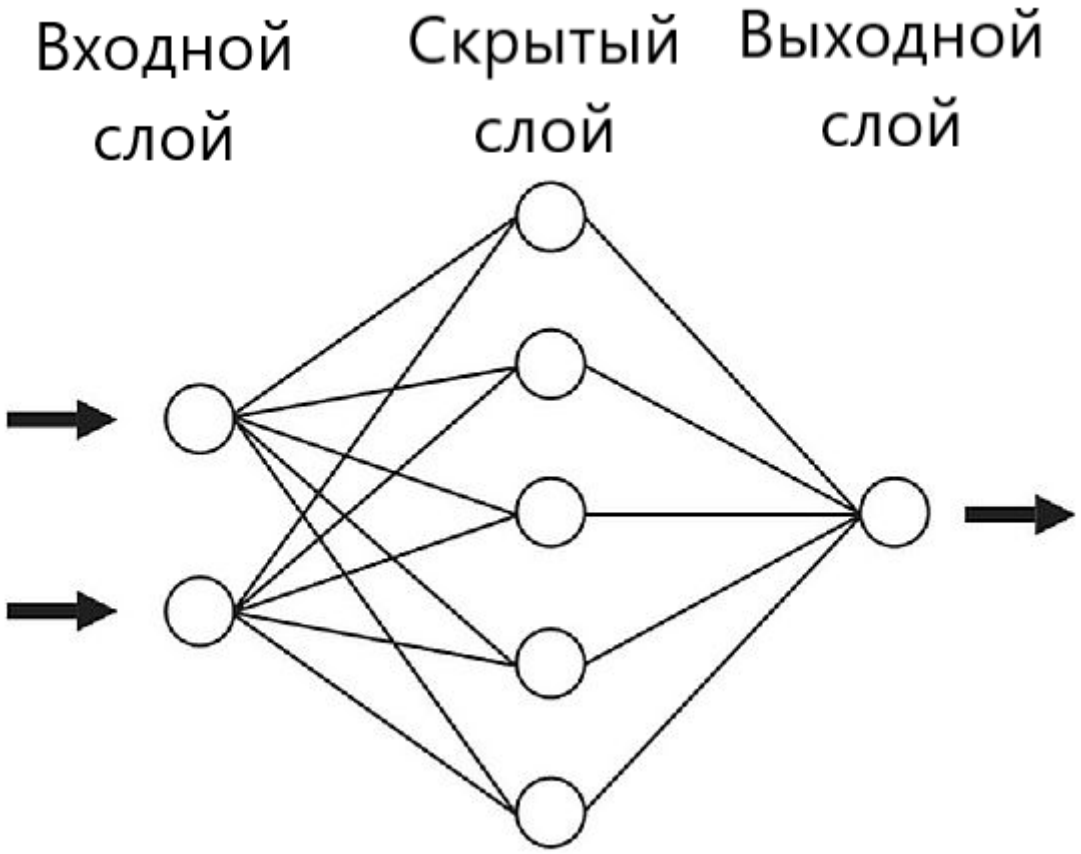

Рисунок 1 – Общая модель нейросети

Существует три вида слоев: входной, скрытый, которых может быть несколько, и выходной. В каждой ячейке, нейроне, входного слоя находится значение от 0 до 1. Связи, соединяющие нейроны также имеют числовое значение, называющееся весом. Значение веса неизвестно. Значения нейронов каждого следующего слоя является суммой произведений значения нейронов предыдущего слоя и веса связи, соединяющих их.

Процесс обучения нейросети представляет из себя поиск такого значение веса, чтобы получить в выходном слое желаемый результат.

Существует два типа обучения нейросетей: с учителем и без него.

Обучении с учителем на вход подается выборка обучающих примеров, в которых известен верный ответ, после чего нейросеть подбирает такой алгоритм, вывод которого приведет которое приведет к тому же результату, что был указан.

Данный метод может быть использован, лишь в том случае, если у решаемой задачи есть единственно верный ответ.

Процесс обучения без учителя подразумевает обучение нейросети без участия человека в нем. В отличие от обучения с учителем, в данном типе обучения нет явных указаний, как поступать в данной ситуации. Успешность алгоритма определяется по общей оценки всех действий, выполненных в процессе решения задачи. Корректировка весовых коэффициентов происходит без участия человека.

Так как в случае разработки модели поведения для мобов нет единственно верного решения задачи, в данной работе будет использоваться тип обучения без учителя.

Подразделяют три алгоритма обучения нейросетей: метод обратного распространения, метод упругого распространения и генетический алгоритм.

Метод обратного распространения ошибки: данный метод подразумевает, что после каждого прохода, происходит обратное распространения, регулирующие п коэффициенты веса. На основе этих данных строится график зависимости веса и отклонения от желаемого результата. Обучение нейросети происходит за счет расчета глобального минимума для каждого коэффициента веса на графике.

Недостатком данного метода является то, что необходимо изменять коэффициенты веса малыми шагами, в обратном случае можно пропустить глобальный минимум.

Метод упругого распространения: данный метод был разработан как альтернатива методу обратного распространения ошибки. Так как прошлый

метод требует слишком много времени, данный метод был ориентирован на скорость.

При методе упругого распространения корректировка весовых коэффициентов происходит крупными шагами для того, чтобы проверить пройден ли локальный минимум используется производные частного случая. Если производная меняет знак, знак, то локальный минимум был упущен, следовательно нужно вернуться к предыдущему значению веса и уменьшить величину изменения.

Если производная меняет свой знак с положительного на отрицательный, то это говорит о росте ошибки. Следовательно коэффициент веса требуется уменьшить. В противоположной ситуации, вес нужно увеличить.

Если изменений в знаке не происходит, цикл повторяется.

Положительной стороной данного метода является то, что требуемый результат удается получить в гораздо быстрее, чем при методе обратного распространения ошибки.

Генетический алгоритм обучения: данный подход к проблеме обучения нейросети основан по принципу эволюции живых существ. Происходит произвольное изменение коэффициентов веса, если с новыми показателями нейросеть имеет лучший результат, то новые значения сохраняются, в ином случае возвращаются старые значения.

Цикл продолжается до тех пор, пока улучшение результата не станет невозможным.

В данной работе будет использоваться генетический алгоритм обучения.

1.3 Вывод

В данной главе был проведен анализ основных технологических решений, применяемых для разработки моделей поведения мобов в компьютерных играх, их положительные и отрицательные стороны, а также существующие примеры

использования нейросетей в данной области. По итогу данного анализа, можно сделать вывод, что несмотря на сложность реализации, использование нейросетей, для формирования моделей поведения в компьютерных играх более перспективно, чем использование скриптов.

2 Определение требований

2.1 Функциональные требования

На этапе проектирования системы были выявлены следующие функциональные требования:

– нейросеть должна создавать новые модели поведения;

– в системе должно иметься хранилище данных, в которой хранятся модели поведения;

– система должна оценивать эффективность модели поведения;

– коэффициент эффективности вычисляется из уничтожения противников, союзников и собственного выживания;

– параметр уничтожения противников имеется более приоритетным, чем собственное выживание;

– система должна сравнивать новые со старыми моделями поведения;

– система должна удалять старые модели поведения;

– каждая модель поведения должна обеспечивать поиск противника, его уничтожения, а также предотвращение собственного уничтожения;

– в процессе обучения нейросети в симуляции должно иметься не менее двенадцати NPC, разделенных на две команды;

– команды должны иметь равное количество NPC;

– одна из команд будет получать новые модели поведения из нейросети, в то время как другая будет иметь изначальную модель поведения;

– время симуляции не должно превышать 5 минут;

– система должна иметь возможность ускорения симуляции;

– система должна иметь возможность остановки симуляции;

– пользователь должен иметь возможность наблюдения за симуляцией;

– система должна осуществляться запись коэффициентов эффективности в отчет.

2.2 Нефункциональные требования

На этапе проектирования были выявлены следующие нефункциональные требования:

- разработка должна вестись на windows 7, 8, 8.1, 10;
- разработка должна вестись на платформе Unity;
- разработка должна вестись на языке c#.

3 ПРОЕКТИРОВАНИЕ

3.1 Архитектура предлагаемого решения

При разработке нейросети для создания моделей поведения мобов в компьютерных играх, была сформирована следующая диаграмма компонентов для отображения взаимодействия логических частей системы, представленная ниже, рисунок 2.

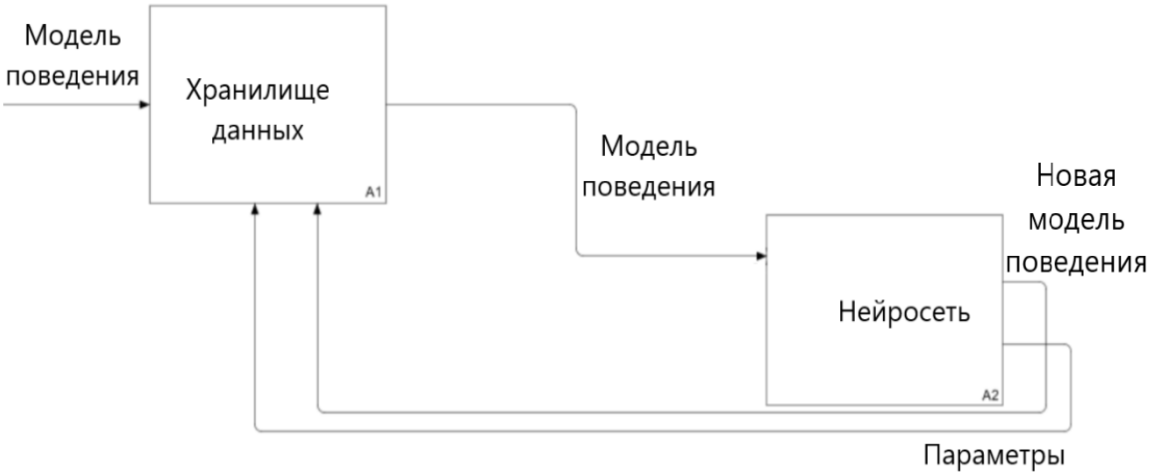

Рисунок 2 – Диаграмма компонентов проектируемой системы

Разрабатываемая система будет состоять из двух взаимосвязанных объектов: хранилище данных и нейросеть.

Хранилище данных в качестве входных данных получит первоначальную модель поведения. Также из нейросети будут получены модель поведения, а также параметры (количество уничтоженных противников, смертей товарищей по команде и убийство членов своей команды), полученные в ходе тестирования модели поведения. Функциями хранилища данных является хранилище модели поведения, сравнивания их между собой посредством коэффициента эффективности и удаление наименее продуктивных. Из хранилища данных в нейросеть будет отправляться наиболее эффективная модель поведения.

Нейросеть будет получать из хранилище данных модель поведения. После чего будет сформирована новая модель и в ходе тестирования вычислены ее

коэффициенты эффективности. Далее полученные данные будут отправлены в хранилище данных.

3.2 Архитектура нейросети

Нейросеть, разработанная для решения задачи, поставленной перед этой работой, имеет структуру, представленную на рисунке ниже (рисунок 3).

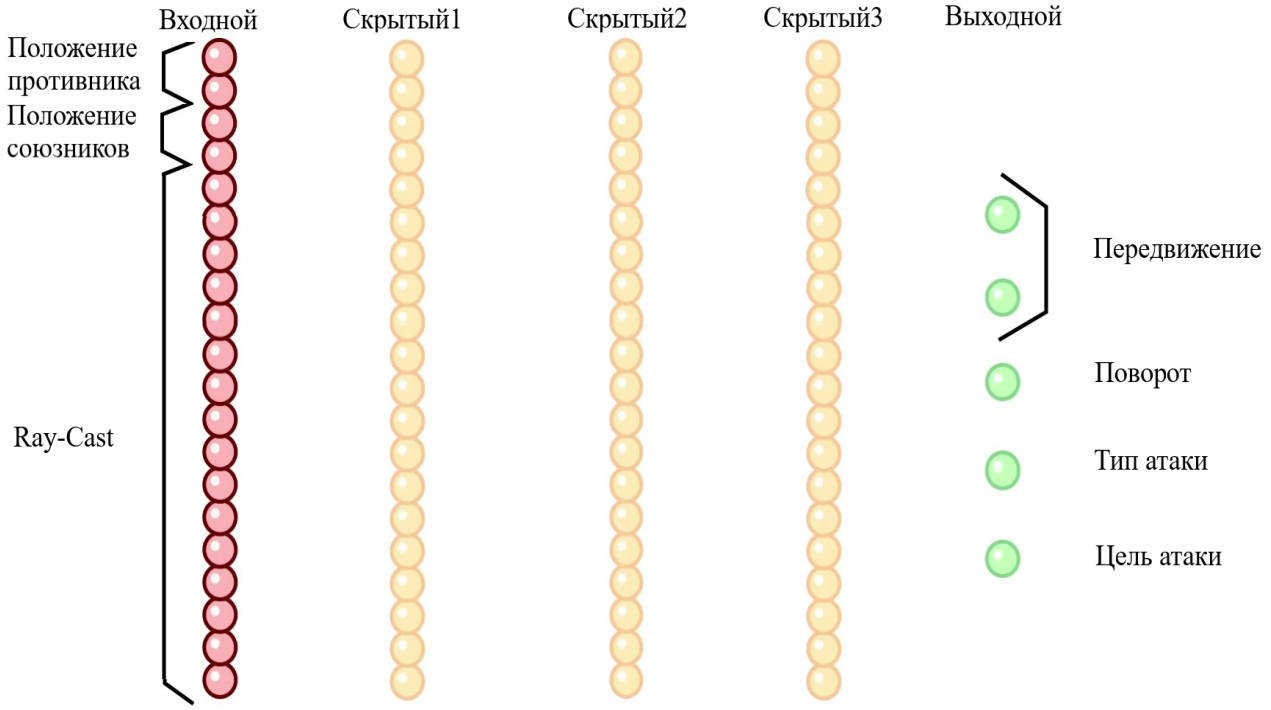

Рисунок 3 – Схема разработанной нейросети

В качестве входных данных нейросеть получает модель поведения, сама же модель поведения получает на вход местоположение противников, союзников и препятствий вокруг моба. Следовательно, на вход нейросети подаются: расстояние до ближайшего противника, угол поворота к нему, расстояние до ближайшего союзника, угол поворота к нему, а также окружающая обстановка, полученная посредством Ray-Cast. Ray-Cast – метод изучения окружающих объектов посредством запуска лучей. Таким образом, имеется двадцать нейронов на входе.

Нейрон смещения в данном случае не нужен.

Опытным путем было установлено, что нейросеть обучается наиболее эффективно при трех скрытых слоях.

В качестве выходных данных имеются два нейрона, отвечающих за перемещение по оси x и y, нейроны, отвечающие за поворот, тип атаки и цель атаки.

3.3 Алгоритм работы системы

В разрабатываемой системе проблемы необходимости большого объема входных данных и излишней сложности игры решены следующим образом: на входе будет иметься готовая модель поведения, что позволит значительно сократить объем входных данных, будут проведены симуляция боя, после которого нейросеть формирует новую модель поведения, но применена она будет только для одной команды. Далее процесс повторяется. В ходе симуляции мобы будут получать очки за свои действия. Данные очки позволят сравнивать модели поведения между собой и выделять наиболее эффективные. Наименее эффективные модели поведения будут удаляться, что не позволит накапливаться ненужным данным в базе данных. Блок схема, описывающая алгоритм представлена ниже на рисунке 4.

ю

Рисунок 4 – Алгоритм решения задачи

3.4 Описание данных

В данной работе входными данными является модель поведения. Модель поведения представляет из себя блок программного кода на языке c#, следовательно будет содержаться в файлах с расширением (.cs). Выходными и промежуточных данными выступают модели поведения и коэффициенты эффективности. Коэффициенты эффективности, это числовое значение,

обозначающие количество балов, полученных за выполнение мобов определенных действий в ходе тестирования. Будут передаваться в качестве текстового файла (.txt).

Также будет вестись отчет по эффективности моделей поведения. Данный отчет будет представлять следующую структуру (рисунок 5).

Убитые противники	Смертей Союзников	Убитые союзники
24	23	6
24	22	6
23	21	6
24	23	6
24	23	6
22	22	5
21	23	6
22	21	6
23	21	5
21	22	6
22	22	5

Рисунок 5 – Схема разработанной базы данных отчет

4 Реализация

4.1 Обучение нейросети

Обучение нейросети будет происходить по методу мутации. Суть этого метода заключается в случайном изменении коэффициентов веса, сохранении удачных комбинаций, и отката на прошлую версию в случае неудачи.

4.2 Модель поведения мобов

Модель поведения мобов должна обладать следующими функциями: перемещение, поиск противника, поворот к нему, атака, способность отличить врагов от союзников.

Поиск противника будет осуществляться с помощью ray-cast. Данный метод подразумевает проведения лучей от заданного объекта. Лучи будут отправляться от моба к объектам вокруг. После чего будет приниматься решение, нужно ли стрелять или нет. С помощью этого метода также будут определяться границы карты.

Предусмотрено два способа атаки: выстрел и бросок гранаты. При взрыве гранаты, все находящиеся в радиусе взрыва мобы получают урон. Этот способ атаки может поставить бота перед выбором, следует ли использовать гранату против группы противников, даже если взрыв может задеть самого моба и его союзников.

Конечным результатом данной работы является модель поведения, эффективней той, что была подана на вход. Эффективность модели поведения будет судиться по следующим параметрам: количество убитых противников, количество смертей союзников и убийство союзников. Получение этих трех параметров обеспечено столкновениями между двумя командами противников, при этом только у одной из команд модель поведения обновляться, в то время

как у второй сохранилась первоначальная модель поведения. Всего в командах по шесть участников. Время возрождения моба одна минута. Время проведения симуляции пять минут.

Числовое значение коэффициента эффективности определяются по формуле (1).

$$Eff = (Fr - 2 * D - 5 * FrF + 210) / 240 * 100\% \quad (1)$$

Где Eff – коэффициент эффективности,

Fr – количество убитых противников,

D – количество смертей союзников,

FrF – убитые союзники.

Модель поведения мобов, построенная посредством нейросети, считается приемлемой, если коэффициент эффективности модели поведения равен 95%. Если значение коэффициента эффективности меньше, то модель поведения изменяется нейросетью. Для этого будет использован генетический алгоритм. Данный алгоритм подразумевает, что в модель поведения будут вноситься произвольные изменения и из полученных будет выбрана наиболее эффективная. Коэффициент эффективности начальной модели поведения равен 66%. Данный коэффициент был определен экспериментально, в ходе проведения тестового боя.

Некоторые из параметров, полученные в ходе работы представлены в ниже (таблица 2).

Таблица 2 – Параметры моделей поведения

Убийство противника	24	23	18	18	0	0	15	20	23	19	24
Смерть товарищей по команде	23	13	13	12	5	3	9	11	9	1	2
Убийство членов своей команды	6	6	4	4	0	0	0	0	0	1	1
Коэффициент эффективности	66%	74%	76%	77%	83%	85%	86%	87%	89%	90%	95%

График изменения коэффициента эффективности представлен далее (рисунок 6).

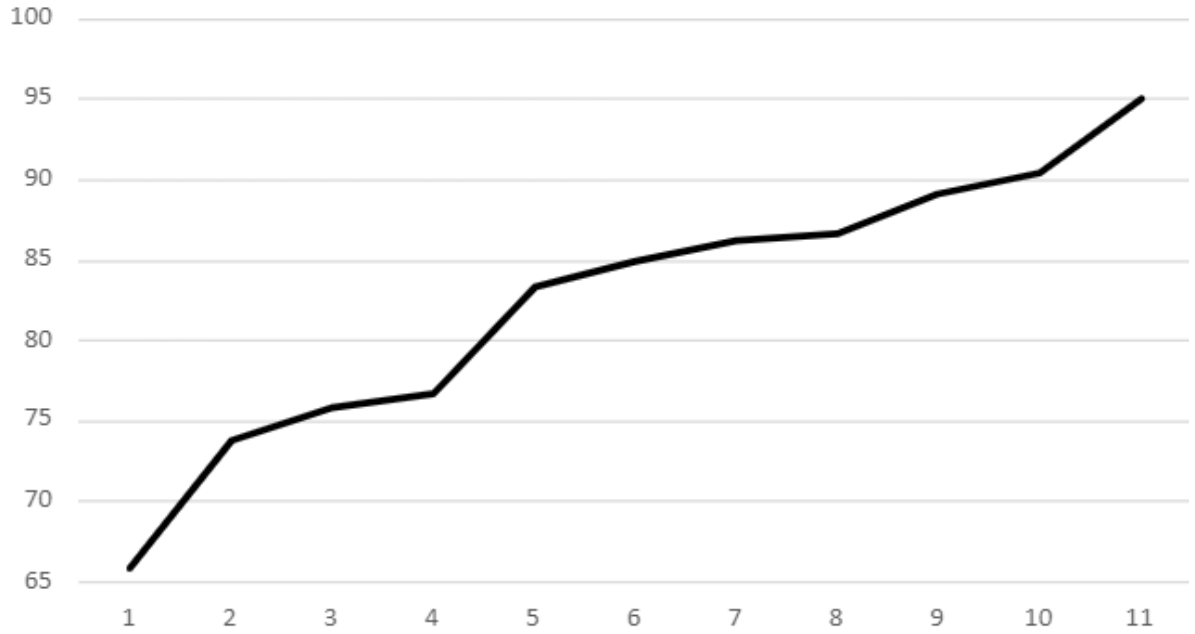

Рисунок 6 – Изменение коэффициента эффективности

5 ТЕСТИРОВАНИЕ

Для тестирования модели поведения были выбраны следующие виды тестирования:

– юзабилити тестирование;

– функциональное тестирование.

5.1 Методология тестирования

Юзабилити тестирование – это метод оценки интерфейса со стороны удобства и эффективности его использования. Для проведение данного метода тестирования необходимо привлечь представителей целевой аудитории программного продукта.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

5.2 Проведение процедуры тестирования

Проверка успешности проведенной работы выполнена путем проведения тестовых боя между командой с изначальной моделью поведения и конечной. В ходе тестирования были получены следующие результаты, представленные ниже (таблица 3).

Таблица 3 – Сравнение параметров моделей поведения

	Первоначальная	**Конечная**
Убийство противника	24	24
Смерть	23	2

Убийство союзника	6	1
Коэффициент эффективности	66%	95%

Как можно видеть из результатов, значительно уменьшилось количество смертей и случаев убийства союзников.

ЗАКЛЮЧЕНИЕ

В представленной работе были разобраны все основные темы, связанные с разработкой модели поведения мобов в компьютерной игре посредством нейросети. Были выделены причины, по которым требовалась разработка собственной нейросети. Обозначены задачи, которые должны быть решены для поставленной цели. Была спроектирована, разработана и обучена нейросеть, для выполнения поставленной задачи.

На стадии разработки использовалось свободное программное обеспечение: Visual Studio и Unity. В качестве языка программирования использовался C#.

По итогам работы, можно сделать следующие выводы:

– рассмотрены существующие программные решения, реализующие нейросеть для моделирования поведения мобов;

– разработана собственная нейросеть способная моделировать поведение мобов;

– нейросеть была обучена;

– полученная модель поведения была оценена по заданным параметрам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Смольников, П. AlphaStar – новая система искусственного интеллекта для StarCraft II от DeepMind (полный перевод). – Текст : электронный / П. Смольников // ХАБР : [сайт]. – URL: https://habr.com/ru/post/437486/ (дата обращения: 12.05.2022).

2. DeepMind : [официальный сайт] / DeepMind. – США, 2022 – . – URL: https://www.deepmind.com/research/highlighted-research/alphago (дата обращения: 12.05.2022). – Текст. Изображение: электронные.

3. Медведева, Я. Как работает бот OpenAI Five – Чемпионат. – Текст. Изображение (неподвижное; двумерное) : электронные / Я. Медведева // Чемпионат: [сайт]. – URL: https://www.championat.com/cybersport/article-3738019-kak-rabotaet-bot-openai-five.html?utm_source=copypaste (дата обращения: 12.05.2022).

4. AlphaGo против Ли Седоля: итоги и оценки профессиональных игроков в го / atomlib // ХАБР : [сайт]. – URL: https://habr.com/ru/post/391747/ (дата обращения: 12.05.2022).

5. Медведева, Я. Матч OpenAI против OG,Dota 2, хайлайты – Чемпионат. – Текст. Изображение (неподвижное; двумерное) : электронные / Я. Медведева // Чемпионат : [сайт]. – URL: https://www.championat.com/cybersport/article-3730041-match-openai-protiv-og-dota-2-hajlajty.html (дата обращения: 12.05.2022).

6. Паласиос, Х. Unity 5.x Программирование искусственного интеллекта в играх / Х. Паласиос. – Москва: ДМК Пресс, 2017. − 272 с.: ил.

7. Вагнер, Б. М. C# Эффективное программирование / Б. М. Вагнер. – Москва: Издательство ЛОРИ, 2017. - 320 с.

8. Мартин, Р. С. Чистый код: создание, анализ и рефакторинг / Р. Мартин. – Москва: Издательство Питер, 2018. – 464 с.

9. Хокинг, Д. Р. Unity в действии / Д. Р. Хокинг. – Санкт-Петербург: Издательство Питер, 2016. – 704 с.

10. Дашко, Ю. В. Основы разработки компьютерных игр / Ю. В. Дашко, А. А. Заика. – Москва: Форум, 2009. – 350 с.

11. Дикинсон, К. Оптимизация игр в Unity 5 / К. Дикинсон; пер. с англ. Рагимова Р. Н. – Москва: ДМК Пресс, 2017. – 306 с.

12. Линовес, Д. Виртуальна реальность в Unity / Д. Линовес. – Москва: ДМК Пресс, 2016. – 316 с.

13. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджет. – Москва : Вильямс, 2012. – 272 с.

14. Мэннинг, Дж. Unity для разработчика. Мобильные мультиплатформенные игры / Дж. Мэннинг, П. Батфилд-Эддисон. – Санкт-Петербург : Питер, 2018. – 304 с.

15. Скиена, С. Алгоритмы. Руководство по разработке / С. Скиена; пер. с англ. – Санкт-Петербург.: БХВ-Петербург, 2011. – 720 с.

16. Стиллмен, Э. Изучаем C# / Э. Стиллмен, Дж. Грин. – Санкт-Петербург : Питер, 2014. – 816 с.

17. Торн, А. Основы анимации в Unity / А. Торн; пер. с англ. Р. Рагимова. – Москва : ДМК Пресс, 2016. – 176 с.

ПРИЛОЖЕНИЕ А
Код нейросети

Листинг 1 – Инициализация функций

```
private int[] layers;//layers
private float[][] neurons;//neurons
private float[][] biases;//biasses
private float[][][] weights;//weights
private int[] activations;//layers
public float fitness = 0;//fitness
public NeuralNetwork(int[] layers)
{
  this.layers = new int[layers.Length];
  for (int i = 0; i < layers.Length; i++)
     this.layers[i] = layers[i];
  InitNeurons();
  InitBiases();
  InitWeights();
}
```

Листинг 2 – Массив веса

```
private void InitWeights()
{
 List<float[][]> weightsList = new List<float[][]>();
 for (int i = 1; i < layers.Length; i++)
 {
   List<float[]> layerWeightsList = new List<float[]>();
   int neuronsInPreviousLayer = layers[i - 1];
   for (int j = 0; j < neurons[i].Length; j++)
   {
     float[] neuronWeights = new float[neuronsInPreviousLayer];
     for (int k = 0; k < neuronsInPreviousLayer; k++)
     {
       neuronWeights[k] = UnityEngine.Random.Range(-0.5f, 0.5f);
     }
     layerWeightsList.Add(neuronWeights);
   }
   weightsList.Add(layerWeightsList.ToArray());
 }
 weights = weightsList.ToArray();
```

```
}
```


Листинг 3 – Выделение памяти для нейронов

```
private void InitNeurons()
{
 List<float[]> neuronsList = new List<float[]>();
 for (int i = 0; i < layers.Length; i++)
 {
   neuronsList.Add(new float[layers[i]]);
 }
 neurons = neuronsList.ToArray();
}
```

Листинг 4 – Расчет нейрона

```
public float[] FeedForward(float[] inputs)
{
 for (int i = 0; i < inputs.Length; i++)
 {
   neurons[0][i] = inputs[i];
 }
 for (int i = 1; i < layers.Length; i++)
 {
   int layer = i - 1;
   for (int j = 0; j < neurons[i].Length; j++)
   {
     float value = 0f;
     for (int k = 0; k < neurons[i - 1].Length; k++)
     {
       value += weights[i - 1][j][k] * neurons[i - 1][k];
     }
   neurons[i][j] = activate(value + biases[i][j]);
   }
 }
 return neurons[neurons.Length - 1];
}
```

Листинг 5 – Инициализация и заполнение смещений

```
    private void InitBiases()
    {
        List<float[]> biasList = new List<float[]>();
        for (int i = 0; i < layers.Length; i++)
        {
            float[] bias = new float[layers[i]];
            for (int j = 0; j < layers[i]; j++)
            {
                bias[j] = UnityEngine.Random.Range(-0.5f, 0.5f);
            }
            biasList.Add(bias);
        }
        biases = biasList.ToArray();
}
```

ПРИЛОЖЕНИЕ Б
Генетический алгоритм

Листинг 6 – Сохранение значений нейронов и веса

```
neural network.
public void Load(string path)
{
 TextReader tr = new StreamReader(path);
 int NumberOfLines = (int)new FileInfo(path).Length;
 string[] ListLines = new string[NumberOfLines];
 int index = 1;
 for (int i = 1; i < NumberOfLines; i++)
 {
   ListLines[i] = tr.ReadLine();
 }
 tr.Close();
 if (new FileInfo(path).Length > 0)
 {
   for (int i = 0; i < biases.Length; i++)
   {
     for (int j = 0; j < biases[i].Length; j++)
     {
       biases[i][j] = float.Parse(ListLines[index]);
       index++;
     }
   }
   for (int i = 0; i < weights.Length; i++)
   {
     for (int j = 0; j < weights[i].Length; j++)
     {
       for (int k = 0; k < weights[i][j].Length; k++)
       {
         weights[i][j][k] = float.Parse(ListLines[index]);
         index++;
       }
     }
   }
 }
}
```

Листинг 7 – Генетический алгоритм

```
public void Mutate(int chance, float val)
{
 for (int i = 0; i < biases.Length; i++)
 {
   for (int j = 0; j < biases[i].Length; j++)
   {
          biases[i][j] = (UnityEngine.Random.Range(0f, chance) <= 5) ?
biases[i][j] += UnityEngine.Random.Range(-val, val) : biases[i][j];
   }
 }

 for (int i = 0; i < weights.Length; i++)
 {
   for (int j = 0; j < weights[i].Length; j++)
   {
     for (int k = 0; k < weights[i][j].Length; k++)
     {
          weights[i][j][k] = (UnityEngine.Random.Range(0f, chance) <= 5) ?
weights[i][j][k] += UnityEngine.Random.Range(-val, val) : weights[i] [j][k];
     }
   }
 }
}
```

Листинг 8 – Загрузка прошлых значений коэффициентов веса

```
public void Load(string path)
    {
        TextReader tr = new StreamReader(path);
        int NumberOfLines = (int)new FileInfo(path).Length;
        string[] ListLines = new string[NumberOfLines];
        int index = 1;
        for (int i = 1; i < NumberOfLines; i++)
        {
            ListLines[i] = tr.ReadLine();
        }
        tr.Close();
```

```
            if (new FileInfo(path).Length > 0)
            {
                for (int i = 0; i < biases.Length; i++)
                {
                    for (int j = 0; j < biases[i].Length; j++)
                    {
                        biases[i][j] = float.Parse(ListLines[index]);
                        index++;
                    }
                }
                for (int i = 0; i < weights.Length; i++)
                {
                    for (int j = 0; j < weights[i].Length; j++)
                    {
                        for (int k = 0; k < weights[i][j].Length; k++)
                        {
                            weights[i][j][k] = float.Parse(ListLines[index]); ;
                            index++;
                        }
                    }
                }
    }
```

Листинг 9 – Мутация

```
    public void Mutate(int chance, float val)
        {
            for (int i = 0; i < biases.Length; i++)
            {
                for (int j = 0; j < biases[i].Length; j++)
                {
                      biases[i][j] = (UnityEngine.Random.Range(0f, chance) <= 5)
? biases[i][j] += UnityEngine.Random.Range(-val, val) : biases[i][j];
                }
            }
            for (int i = 0; i < weights.Length; i++)
            {
```

```
        for (int j = 0; j < weights[i].Length; j++)
                    {
                        for (int k = 0; k < weights[i][j].Length; k++)
      {
                weights[i][j][k] = (UnityEngine.Random.Range(0f, chance) <= 5) ?
weights[i][j][k] += UnityEngine.Random.Range(-val, val) : weights[i][j][k];
                        }
                    }
                }
      }
```

Листинг 10 – Сравнение эффективности

```
    public int CompareTo(NeuralNetwork other
    {
        if (other == null) return 1;

        if (fitness > other.fitness)
            return 1;
        else if (fitness < other.fitness)
            return -1;
        else
            return 0;
    }
```

ПРИЛОЖЕНИЕ В
МОДЕЛЬ ПОВЕДЕНИЯ МОБОВ

Листинг 11 – Начальная модель поведения

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyAI : MonoBehaviour
{
    public float speed = 5.0f;
    public float obstacleRande = 5.0f;
    public bool _alive = true;
    [SerializeField]
    private GameObject[] _fireballsPrefab;
    private GameObject _fireball;
    private void Start()
    {
        _alive = true;
    }
    private void Update()
    {
        if (_alive)
        {
            transform.Translate(0, 0, speed * Time.deltaTime);
            Ray ray = new Ray(transform.position, transform.forward);
            RaycastHit hit
            }
        }
    }
    public void SetAlive(bool alive)
    {
        _alive = alive;
    }
}
```

Листинг 12 – Конечная модель поведения

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyAI : MonoBehaviour
{
    public float speed = 5.0f;
    public float obstacleRande = 5.0f;
    public bool _alive = true;
    [SerializeField]
    private GameObject[] _fireballsPrefab;
    private GameObject _fireball;
    private void Start()
    {
        _alive = true;
    }
    private void Update()
    {
        if (_alive)
        {
            transform.Translate(0, 0, speed * Time.deltaTime);
            Ray ray = new Ray(transform.position, transform.forward);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
            {
                GameObject hitObject = hit.transform.gameObject;
                if (hitObject.GetComponent<Character>())
                {
                    if (_fireball == null)
                    {
                                        int randFireball = Random.Range(1,
_fireballsPrefab.Length);
                                                                _fireball =
Instantiate(_fireballsPrefab[randFireball]) as GameObject;
                                                    _fireball.transform.position =
transform.TransformPoint(Vector3.forward * 1.5f);
                            _fireball.transform.rotation = transform.rotation;
                    }
                }
```

```
                else if (hit.distance < obstacleRande)
                {
                    float angleRotation = Random.Range(-100, 100);
                    transform.Rotate(0, angleRotation, 0);
                }
            }
        }
    }
    public void SetAlive(bool alive)
    {
        _alive = alive;
    }
}
```

Листинг 13 – Смерть моба

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ReactiveTarget : MonoBehaviour
{
    private EnemyAI _enemyAI;
    private void Start()
    {
        _enemyAI = GetComponent<EnemyAI>();
    }
    public void ReactToHit()
    {
        if (_enemyAI != null)
            _enemyAI.SetAlive(false);
        StartCoroutine(DieCoroutine(3));
    }
    private IEnumerator DieCoroutine(float waitSecond)
    {
        this.transform.Rotate(45, 0, 0);
        yield return new WaitForSeconds(waitSecond);
        Destroy(this.transform.gameObject);
    }
}
```

Листинг 14 – Возрождение противников

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyController : MonoBehaviour
{
    [SerializeField]
    private GameObject[] _enemyPrefab;
    private GameObject _enemy;
    private void Update()
    {
        if(_enemy < 6)
        {
            int randEnemy = Random.Range(1, _enemyPrefab.Length);
            _enemy = Instantiate(_enemyPrefab[randEnemy]) as GameObject;
            _enemy.transform.position = new Vector3(0, 3, 0);
            float angle = Random.Range(0, 360);
            _enemy.transform.Rotate(0, angle, 0);
        }
    }
}
```

Листинг 15 – Возрождение союзников

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CharacterController : MonoBehaviour
{
    [SerializeField]
    private GameObject[] _characterPrefab;
    private GameObject _character;
    private void Update()
    {
        if(_character < 0)
        {
```

```
        int randCharacter = Random.Range(1, _characterPrefab.Length);
                    _character = Instantiate(_characterPrefab[randCharacter]) as
GameObject;
                _character.transform.position = new Vector3(0, 3, 0);
                float angle = Random.Range(0, 360);
                _character.transform.Rotate(0, angle, 0);
            }
        }
    }
```

ПРИЛОЖЕНИЕ Г
ОБЪЕКТЫ

Листинг 16 – Бросок гранаты

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Fireball : MonoBehaviour
{
    public float speed = 10f;
    public int damage = 1;

    private void Update()
    {
        transform.Translate(0,0,speed*Time.deltaTime);
    }
    private void OnTriggerEnter(Collider other)
    {
        Debug.Log(other.name);

        Character character = other.GetComponent<Character>();
        if(character != null)
        {
            character.Hurt(damage);
        }
        Destroy(this.gameObject);
    }
}
```

Листинг 17 – Выстрел

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class laser_script : MonoBehaviour
{
    public GameObject point;
    public GameObject originShooting;
    public GameObject start_laser;
```

```
      void Update()
          {
              RaycastHit hit;
                          if   (Physics.Raycast(originShooting.transform.position,
originShooting.transform.forward, out hit))
              {
                  point.transform.position = hit.point;
                  point.transform.rotation = Quaternion.LookRotation(hit.normal);
                  start_laser.transform.position = hit.point;
                                                  start_laser.transform.rotation    =
Quaternion.LookRotation(hit.normal);
              }
          }
      }
```