

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ
МЕЖИНСТИТУТСКАЯ БАЗОВАЯ КАФЕДРА

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Междисциплинарный проектный практикум»

на тему:

«Разработка компьютерной игры с противником нейросетью»

Выполнил:

Ботвинкин Никита Сергеевич

студент 3 курса

группы ПИЖ-б-о-21-1

направление подготовки 09.03.04

«Программная инженерия»

направленность (профиль)

«Разработка и сопровождение

программного обеспечения»

очной формы обучения

(подпись)

Руководитель проекта:

Свистунов И.В., профессор

межинститутской базовой кафедры

Проект допущен к защите _____

(подпись руководителя)

(дата)

Проект выполнен и

защищен с оценкой _____

Дата защиты _____

Члены комиссии:

зав. межинститутской базовой
кафедрой

Е. Н. Новикова

(подпись)

доцент МИБК

И.В. Свистунов

(подпись)

доцент МИБК

З.М. Альбекова

(подпись)

Ставрополь, 2024 г.

Министерство науки и высшего образования и Российской Федерации Федеральное
государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт Цифрового развития
Кафедра межинститутская базовая

Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) «Разработка и сопровождение программного обеспечения»

ЗАДАНИЕ на курсовую работу

студента Ботвинкина Никиты Сергеевича
(фамилия, имя, отчество)

по дисциплине «Междисциплинарный проектный практикум»

1. Тема работы Разработка компьютерной игры с противником нейросетью
2. Цель: повышение уровня профессиональной подготовки путем углубления и закрепления теоретических знаний и практических навыков, приобретенных в результате изучения дисциплины «Междисциплинарный проектный практикум»; подготовка к самостоятельной разработке программного обеспечения с использованием современных информационных технологий.
3. Задачи:
 - 3.1 Анализ прикладной задачи и методов ее решения. Обоснование выбора средств, технологий и алгоритмов решения прикладной задачи.
 - 3.2 Разработка алгоритмов решения задачи.
 - 3.3 Реализация программного кода
 - 3.4 Отладка и тестирование программного кода
4. Перечень подлежащих разработке вопросов:
 - а) теоретической части: изучение и анализ литературы, постановка условия задачи, выбор и описание методов и библиотек для ее решения и технологий, среды программирования
 - б) проектная часть: проектирование UML-диаграмм, IDEF, интерфейса и пр.
 - в) реализация: описание структуры проекта, описание файлов проекта, разработка программного кода, тестирование программы
5. Исходные данные:
 - а) по литературным источникам: ГОСТы, международные стандарты, программные средства, используемые при разработке программного обеспечения

б) исходные данные, подготовленные для тестирования объекта профессиональной деятельности (информационной системы/приложения/программного продукта)

6. Список рекомендуемой литературы: монографии, диссертации, научные статьи, учебно-методические материалы, ссылки на официальные сайты, содержащие информацию необходимую для решения поставленных в работе задач:

1) Галенкин С. Маркетинг игр [Текст] / С. Галенкин. – Санкт-Петербург: Питер, 2014 – 78 с.

2) История компьютерных игр / [Электронный ресурс] // Wikipedia : [сайт]. – URL: https://ru.wikipedia.org/wiki/История_компьютерных_игр (дата обращения: 31.05.2024).

3) Игровая индустрия: геймдев (gamedev) / [Электронный ресурс] // HSBI : [сайт]. – URL: <https://hsbi.hse.ru/articles/igrovaya-industriya-geymdev/> (дата обращения: 31.05.2024).

4) Платформа Unity для разработки в реальном времени | Движок для 3D, 2D, VR и AR / [Электронный ресурс] // unity : [сайт]. – URL: <https://unity.com/ru> (дата обращения: 31.05.2024).

7. Контрольные сроки представления отдельных разделов курсового проекта:

25 % - представление первого раздела «07» марта 2024 г.

50 % - представление второго раздела «15» апреля 2024 г.

75 % - представление третьего раздела «29» апреля 2024 г.

100 % - представление работы на отзыв «10» мая 2024 г.

8. Срок защиты студентом курсового проекта «23» мая 2024 г.

Дата выдачи задания «05» февраля 2024 г.

Руководитель курсовой работы

К. технических н., доцент

(учебная степень, звание)

(личная подпись)

И. В. Свистунов

(инициалы, фамилия)

Задание принял к исполнению студент очной формы обучения 3 курса
группы ПИЖ-б-о-21-1

(личная подпись)

Н.С. Ботвинкин

(инициалы, фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.....	8
1.1 Предметная область	8
1.2 Анализ конкурентов	9
1.3 Техническое задание	10
1.3.1 Требования к функциональным характеристикам программы.....	11
1.3.2 Требования к техническим характеристикам программы.....	11
1.3.3 Требования к программным характеристикам программы.....	12
1.3.4 Этапы разработки программы и график выполнения работ	12
1.4 Технологии для разработки программы.....	13
1.4.1 Среда разработки.....	13
1.4.2 Язык программирования и фреймворк.....	13
1.4.3 Библиотеки.....	14
Выводы	14
2 ПРОЕКТНАЯ ЧАСТЬ	15
2.1 Общий алгоритм работы	15
2.2 Проектирование структуры проекта	16
2.2.1 Скрипты и их функции.....	17
2.2.2 Игровые сцены.....	19
2.3 Проектирование интерфейсов	20
Выводы	22
3 ПРАКТИЧЕСКАЯ ЧАСТЬ	23
3.1 Реализация игры	23
3.1.1 Создание проекта Unity.....	23
3.1.2 Настройка ML-Agents.....	24
3.1.3 Обучение агента	30
3.1.4 Скрипты, функции, ассеты.....	32
3.2 Тестирование	41
3.3 Руководство пользователя	41

3.3.1 Введение	41
3.3.2 Описание операций	42
Выводы	42
ЗАКЛЮЧЕНИЕ	43
СПИСОК ЛИТЕРАТУРЫ	44
ПРИЛОЖЕНИЕ	45
Приложение А	45
Приложение Б.....	59

ВВЕДЕНИЕ

Компьютерные игры стали важной частью современной культуры, предоставляя пользователям увлекательные и разнообразные формы развлечений. В сфере разработки игр создаются проекты, которые позволяют игрокам погружаться в виртуальные миры, испытывать захватывающие приключения и соревноваться с другими игроками. Однако, несмотря на разнообразие игр для конечных пользователей, существует дефицит специализированных решений, которые помогают разработчикам эффективно интегрировать продвинутых противников с использованием нейросетей.

Объект исследования: процессы разработки компьютерных игр с противником, управляемым нейросетью.

Предмет исследования: разработка компьютерной игры с противником на базе нейросети для платформы PC.

Цель работы: разработка компьютерной игры, в которой противник управляется нейросетью, что сделает игровой процесс более сложным и интересным для игроков.

Для достижения этой цели необходимо решить следующие задачи:

1. Определить требования к функциональности и дизайну игры на основе анализа потребностей игроков и существующих решений.
2. Разработать архитектуру и функциональные модули игры, включая интеграцию нейросетевого противника.
3. Реализовать игру с использованием среды разработки Unity и языка программирования C#.
4. Обучить нейросеть для управления противником и провести тестирование игры на реальных пользователях, анализируя полученные данные для дальнейшего улучшения продукта. Методы исследования

Методы исследования

В процессе разработки будут использованы следующие методы:

1. Анализ. Анализ существующих игр с умными противниками.

2. Проектирование и моделирование. Создание прототипа игры, разработка архитектуры и дизайна пользовательского интерфейса, а также нейросетевого противника.

3. Программирование. Реализация функциональности игры с использованием Unity и языка C#, а также обучение и интеграция нейросети.

4. Тестирование. Проведение функционального и пользовательского тестирования игры, сбор обратной связи и корректировка функциональности на основе полученных данных.

Таким образом, данная курсовая работа направлена на создание компьютерной игры, в которой противник управляется нейросетью, что позволит сделать игровой процесс более динамичным и интересным. Разработка и внедрение такой игры будет способствовать развитию новых подходов в геймдизайне, повышению удовлетворенности игроков и внедрению современных технологий в игровую индустрию.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В теоретической части работы рассмотрены основные аспекты разработки компьютерной игры с нейросетевым противником. Это включает анализ требований пользователей, выбор технических средств, описание функциональных и технических характеристик игры, а также этапы разработки и график выполнения работ. Ключевыми моментами являются обоснование выбора технологий, определение этапов разработки и разработка стратегии обеспечения качества игры.

1.1 Предметная область

Разработка компьютерной игры с нейросетевым противником требует глубокого понимания предметной области создания интерактивного программного обеспечения. Она включает анализ требований игроков, выбор соответствующих технологий, детальное описание функциональных и технических характеристик игры, а также этапы разработки и составление графика выполнения проекта.

Особое внимание уделяется обоснованию выбора используемых технологий, определению ключевых этапов разработки и созданию стратегии для обеспечения высокого качества игры. Важными элементами разработки являются создание продвинутого ИИ противника с использованием нейросетевых алгоритмов, а также адаптация игрового процесса для обеспечения оптимального опыта игрока.

Предметная область разработки компьютерных игр с нейросетевыми противниками предоставляет уникальные возможности для создания более динамичных и сложных игровых сценариев, что повышает интерес и вовлеченность пользователей.

1.2 Анализ конкурентов

Анализ искусственного интеллекта противников в различных других играх показал, что в основном при проектировании ИИ врагов, используются такие подходы, как деревья поведения (Behavior Trees), поисковые алгоритмы, системы планирования (Goal-Oriented Action Planning, GOAP), системы конечных автоматов (Finite State Machines, FSM).

Однако некоторые студии используют машинное обучение с системой наград и наказаний, также известное, как обучение с подкреплением (Reinforcement Learning, RL). Вот несколько примеров игр, которые используют такие подходы:

AlphaGo (DeepMind)

Хотя AlphaGo не является коммерческой видеоигрой, это один из самых известных примеров использования обучения с подкреплением. ИИ AlphaGo обучался играть в игру го, получая награды за победы и штрафы за поражения. AlphaGo стала первой в мире программой, которая выиграла матч без гандикапа у профессионального игрока в го на стандартной доске 19×19 , и эта победа ознаменовала собой важный прорыв в области искусственного интеллекта, так как большинство специалистов по искусственному интеллекту считало, что подобная программа не будет создана ранее 2020 - 2025 годов

Gran Turismo Sport (Sony AI)

Sony AI использовала обучение с подкреплением для создания ИИ-соперника в гоночной игре Gran Turismo Sport. Gran Turismo Sophy – искусственный интеллект (ИИ), который способен выступать в качестве соперников для реальных игроков и выступать на уровне профессиональных автогонщиков. В плане управления автомобилем ИИ отличается глубоким пониманием динамики машины и особенностями гоночных трасс, а также имеет навыки для точного преодоления дистанции. Искусственный интеллект способен принимать решения за доли секунды в ответ на быстро меняющиеся ситуации в гонках, а кроме того, Gran Turismo Sophy соблюдает спортивный

этикет и не позволяет себе некорректного поведения по отношению к соперникам и также реагирует на маневры других участников гонки.

1.3 Техническое задание

Данное техническое задание посвящено разработке компьютерной игры, с противником нейросетью.

Актуальность разработки компьютерной игры с нейросетевым противником обусловлена несколькими ключевыми факторами.

Во-первых, индустрия видеоигр постоянно развивается, и игроки ищут новые, более сложные и интересные игровые сценарии.

Во-вторых, нейросетевые технологии открывают новые возможности для разработки игр, делая искусственный интеллект (ИИ) более реалистичным и интеллектуально способным.

Кроме того, современные технологии и вычислительные мощности делают разработку игр с нейросетевыми противниками более доступной и эффективной.

Наконец, с точки зрения научных и технических исследований, разработка таких игр способствует продвижению знаний и навыков в области искусственного интеллекта и нейросетей.

В совокупности эти факторы делают разработку компьютерной игры с нейросетевым противником актуальной и перспективной задачей, способной удовлетворить современные запросы рынка и способствовать дальнейшему развитию технологий.

Результатом выполнения данного технического задания станет соответствующая заданным требованиям, компьютерная игра, в которой игрок будет противостоять искусственному интеллекту, созданному на основе нейросетевых алгоритмов.

1.3.1 Требования к функциональным характеристикам программы

Разрабатываемая компьютерная игра с противником нейросетью должна обладать следующими функциональными характеристиками:

- 1) Окружение, по которому будет передвигаться персонаж
- 2) Управление персонажем, разработанное под персональный компьютер
- 3) Условия, при которых игрок проигрывает и начинает заново
- 4) Условия, при которых игрок одерживает победу
- 5) Рабочее главное меню

Эти функциональные требования обеспечат создание эффективной и увлекательной компьютерной игры с нейросетевым противником, способной удовлетворить потребности как разработчиков, так и игроков.

1.3.2 Требования к техническим характеристикам программы

Компьютерная игра с противником нейросетью должна соответствовать следующим техническим требованиям:

Платформа и совместимость: игра должна быть разработана для операционных систем Windows, обеспечивая совместимость с версиями не ниже Windows 10. Также игра должна поддерживать различные разрешения экрана и размеры окон для обеспечения комфортного игрового процесса на широком спектре устройств.

Производительность: время загрузки игры не должно превышать 5 секунд на компьютерах с рекомендованными системными требованиями.

Пользовательский интерфейс: интерфейс игры должен быть адаптирован для отображения на экранах с различными разрешениями, обеспечивая удобство использования независимо от типа устройства. Интерфейс должен быть интуитивно понятным и эффективно передавать игровую информацию, обеспечивая комфортное взаимодействие игрока с игровым миром.

Эти технические требования направлены на создание увлекательной и качественной игры с противником на основе нейронных сетей, обеспечивая надежность, безопасность и удобство использования для игроков всех уровней.

1.3.3 Требования к программным характеристикам программы

Программа не имеет специфических требований к программному обеспечению, так как она разработана для функционирования на операционной системе Windows.

1.3.4 Этапы разработки программы и график выполнения работ

1) Понимание требований и концепция

Определить потребности игроков и основные цели игры с учетом противника, управляемого нейросетью. Проработать концепции игры, включая ее основные механики, стиль и общий опыт игрока.

2) Проектирование игровой системы.

Разработать архитектуры и структуру игры с учетом требований и концепции. Определить основные компоненты и взаимодействия в игре, включая поведение нейросетевого противника.

3) Дизайн пользовательского интерфейса

Создать интуитивно понятный интерфейс пользователя, который обеспечивает комфортное взаимодействие с игрой и понимание действий нейросетевого противника.

4) Реализация игровой функциональности

Спроектировать и реализовать основные функции игры, включая механику игрового процесса и поведение противника, управляемого нейросетью.

5) Документирование

Подготовить техническую документацию, описывающую основные компоненты игры, а также API для разработчиков. Создать руководство пользователя, которое поможет игрокам разобраться в игровом процессе и взаимодействии с нейросетевым противником.

б) Тестирование, отладка, улучшение

Провести тестирование игры для выявления ошибок и недочетов в функциональности и интерфейсе. Исправить обнаруженные ошибки и внести улучшения для повышения качества и удовлетворения игрового опыта пользователей.

1.4 Технологии для разработки программы

Чтобы гарантировать оптимальное функционирование компьютерной игры с противником, управляемым нейросетью, необходимо использование следующих программных инструментов:

1.4.1 Среда разработки

Для разработки компьютерной игры с противником, управляемым нейросетью, будут использоваться основные среды разработки: Unity3D и Visual Studio. Unity3D обеспечивает инструменты для создания графики, физики и игровой логики, а Visual Studio служит для написания и отладки кода на C#. Для управления исходным кодом и ресурсами проекта предполагается использование системы контроля версий Git с хостингом на платформе GitHub.

1.4.2 Язык программирования и фреймворк

Для разработки данного проекта используются следующие языки программирования и фреймворки:

Язык программирования: C#

Фреймворк: ML-Agents

Язык программирования C# выбран для разработки основной логики и управления игровым процессом, а также для интеграции с фреймворком ML-Agents, который предоставляет инструменты для обучения и управления нейронными сетями в Unity3D.

1.4.3 Библиотеки

Для разработки игры были использованы следующие библиотеки:

- 1) UnityEditor. Библиотека, предоставляющая инструменты и ресурсы для разработки и настройки игры в среде Unity.
- 2) TMPro. Библиотека для улучшения работы с текстом в Unity, обеспечивающая расширенные возможности форматирования и отображения текста.
- 3) UnityEngine. Основная библиотека Unity, предоставляющая доступ к различным компонентам и функциям движка, таким как управление сценами, объектами, графикой и вводом.
- 4) System.Collections. Стандартная библиотека .NET, включающая основные типы данных и структуры данных, такие как списки, словари и очереди.

Выводы

В разработанном техническом задании для компьютерной игры с противником, управляемым нейросетью, определены основные требования к функциональности, интерфейсу и процессу взаимодействия. Использование Unity3D и языка программирования C# обеспечивает основу для создания игрового контента и интеграции нейросетевых противников, а использование ML-Agents фокусируется на развитии и обучении поведения противников. Все это в совокупности обеспечивает основу для разработки увлекательной и инновационной компьютерной игры с применением передовых технологий и методов искусственного интеллекта.

2 ПРОЕКТНАЯ ЧАСТЬ

В проектной части курсовой работы будет представлено описание выбора языка программирования и среды разработки, а также алгоритм работы приложения. Также будет рассмотрено проектирование игрового контента, включая игровые механики, уровни и интерфейс пользователя, а также интеграция нейросетевого противника в игровой процесс.

2.1 Общий алгоритм работы

При входе в игру открывается главное меню, где пользователю предоставляются две опции: "Играть" и "Выход". При выборе "Играть" запускается уровень, в течение которого игроку предстоит выжить 6 минут. Внизу экрана отображается таймер, а в левом верхнем углу – количество жизней игрока (всего их 3). В течение игрового времени по краям появляются нейросетевые враги, обученные направляться к позиции игрока. Столкновение с врагом приводит к потере жизни. Если игрок теряет все жизни в течение 6 минут, игра завершается. В случае, если игроку удастся продержаться на арене 6 минут, не исчерпав все жизни, он побеждает. После окончания игры, независимо от исхода, пользователь возвращается в главное меню.

Более подробно работа приложения представлена на рисунке 2.1

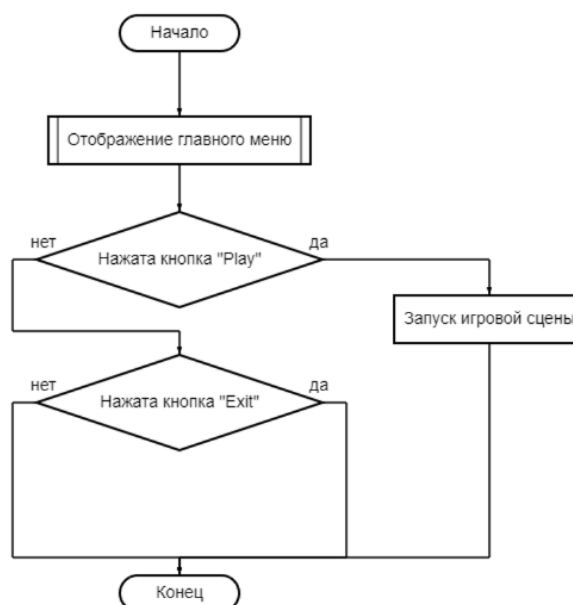


Рисунок 2.1 – Блок-схема работы главного меню

2.2 Проектирование структуры проекта

При проектировании игры, было принято решение разделить все ассеты по соответствующим каталогам:

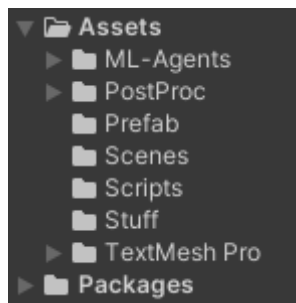


Рисунок 2.2 – Каталоги проекта

- ML-Agents: каталог, содержащий конфигурационные данные, параметры настройки и другую информацию, необходимую для работы алгоритмов машинного обучения или для взаимодействия с ними.

- PostProc: каталог, содержащий профили постобработки. Post Process Profile в Unity содержит параметры и настройки для различных эффектов постобработки, таких как размытие, цветокоррекция, тонирование и другие. Этот файл позволяет вам настроить эффекты постобработки в вашем проекте и применить их к сценам или камерам.

- Prefab: каталог, содержащий в себе префабы. Префабы (Prefabs) в Unity – это предварительно созданные объекты, которые могут быть многократно использованы в проекте.

- Scenes: каталог, содержащий в себе сцены проекта. В моем проекте будет использоваться сцена главного меню, и сцена игры. В Unity, сцена (Scene) представляет собой пространство, в котором располагаются все объекты, эффекты и другие элементы проекта.

- Scripts: каталог, содержащий в себе скрипты. Скрипты (Scripts) в Unity представляют собой программные файлы, написанные на C#. Скрипты используются для создания логики и поведения объектов в проекте Unity.

- Stuff: данный каталог будет содержать в себе остальные типы файлов, которые не были классифицированы. Например, в нем будет храниться обученная модель поведения противника.

- TextMesh Pro: данный каталог содержит ресурсы, связанные с использованием расширенного текстового рендеринга и визуализации, предоставляемого пакетом TextMesh Pro.

2.2.1 Скрипты и их функции

Для реализации игровой логики были созданы 12 скриптов, каждый из которых назначен управлению определенным аспектом игры. Давайте рассмотрим, какие функции выполняются каждым из этих скриптов и какие аспекты игры они обеспечивают.

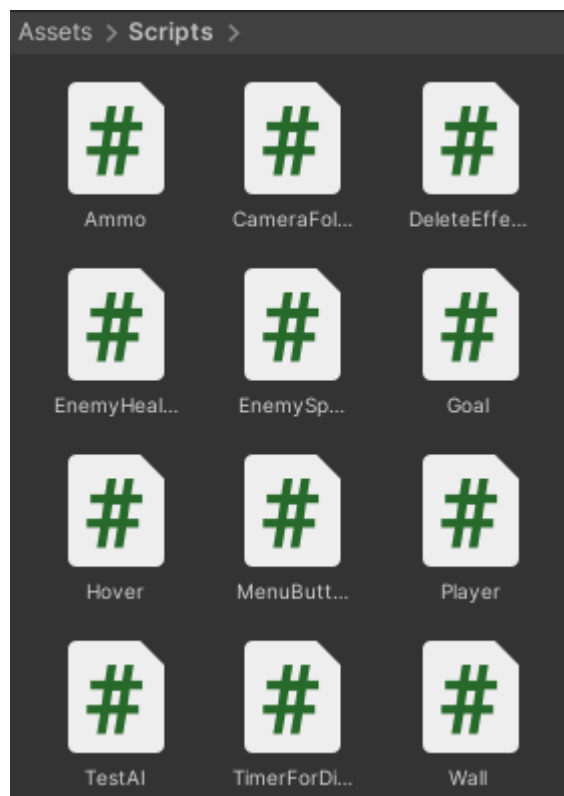


Рисунок 2.3 – Каталог со скриптами

Скрипт «Player» отвечает за управление персонажем. С помощью этого скрипта игрок может перемещаться по игровой арене, атаковать врагов. В этом скрипте описано здоровья игрока, параметры движения и атаки.

Скрипт «Ammo» описывает логику снарядов, которыми стреляет игрок. Также этот скрипт регистрирует попадания снарядов по врагам.

Скрипт «CameraFollow» используется для красивого передвижения камеры за игроком. Также вызывает эффект тряски экрана при соприкосновении с противником

Скрипт «DeleteEffect» предназначен для удаления эффектов стрельбы и убийства противников. Был спроектирован в целях оптимизации

Скрипт «EnemyHealth» необходим для инициирования здоровья противников. При попадании снаряда во врага, показатель его жизни уменьшается на единицу. Изначально он равен 3. Также этот скрипт используется для обнаружения столкновений с игроком.

Скрипт «EnemySpawn» описывает логику появления врагов на арене. С каждой секундой интервал появления врагов уменьшается. Это сделано в целях повышения сложности игры.

Скрипт «Hover» предназначен для увеличения текста под курсором мыши. Используется в главном меню игры.

Скрипт «MenuButtons» отвечает за работоспособность кнопок в главном меню игры.

Скрипт «TimerForDifficulty» изменяет значение игрового таймера. Изначально, значение таймера равно 360 (6 минут).

Скрипты «Wall» и «Goal» используются как тэги для идентификации компонентов

Скрипт «TestAI» используется для обучения нейросети и настройки поведения противников.

Была составлена диаграмма классов, представленная на рисунке 2.4

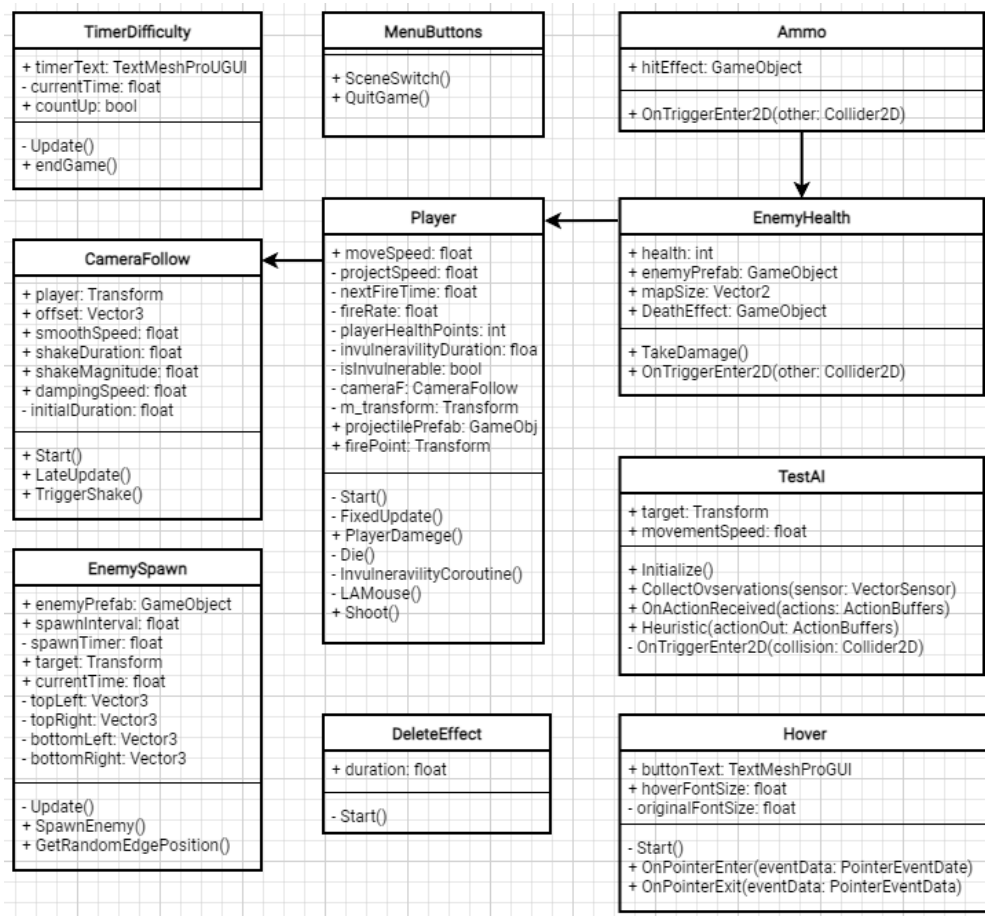


Рисунок 2.4 – Диаграмма классов UML

2.2.2 Игровые сцены

На основе представленной структуры проекта, были разработаны игровые сцены. Первая сцена (MainMenu) представляет собой главное меню игры. Ее структура представлена на рисунке 2.5

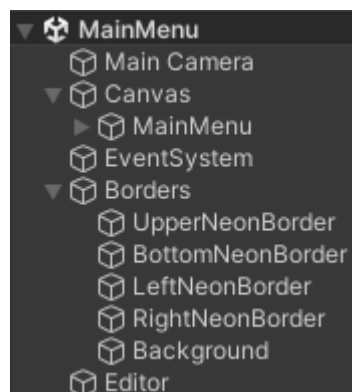


Рисунок 2.5 – Структура сцены главного меню игры

Вторая сцена (GameScene) представляет собой основное пространство для игры. Оно состоит из арены, в рамках которой передвигается игрок и уничтожает врагов. Ее структура представлена на рисунке 2.6

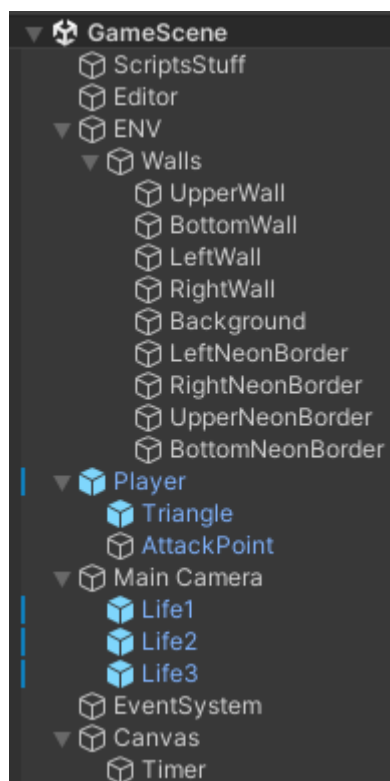


Рисунок 2.6 – Структура сцены игровой арены

2.3 Проектирование интерфейсов

Основываясь на разработанной структуре приложения, были разработаны интерфейсы для двух сцен.

Для сцены главного меню был создан минималистичный интерфейс, включающий две кнопки: «Play» и «Exit». Для достижения визуально привлекательного эффекта использовались различные постобработочные эффекты. Для кнопок был выбран пиксельный шрифт, соответствующий общей стилистике и атмосфере игры. Итоговый результат представлен на рисунке 2.7



Рисунок 2.7 – Интерфейс главного меню игры

Для сцены игровой арены был разработан аналогично минималистичный интерфейс. Неоновое свечение призвано скрыть визуальную простоту изображения. Для удобства игрока таймер размещен в нижней части экрана. Его шрифт такой же пиксельный, как и кнопки главного меню. Жизни игрока расположены в верхнем левом углу и представлены копиями модели игрока, что делает их назначение интуитивно понятным. Итоговый результат разработки интерфейса представлен на рисунке 2.8

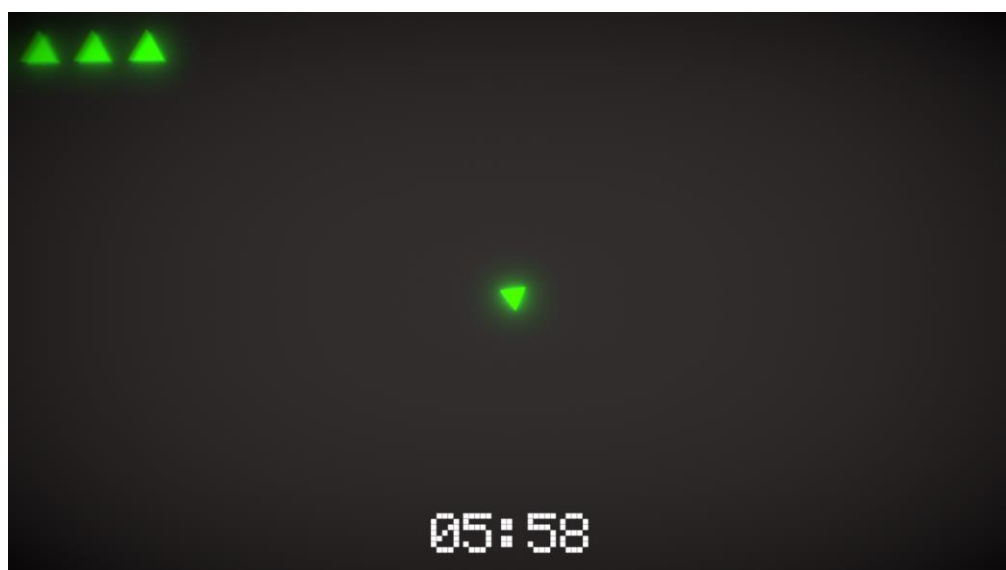


Рисунок 2.8 – Интерфейс игровой арены

Вся игра выполнена в неоновом стиле на темном фоне, что усиливает эффект неоновой свечки. Современная популярность темной темы не только придает игре стильный вид, но и уменьшает нагрузку на глаза.

Выводы

В процессе проектирования игры с противником, управляемым нейросетью, были рассмотрены ключевые аспекты разработки, включая выбор инструментов программирования и среды разработки. Мы также детально описали функциональность скриптов и спроектировали дизайн существующих игровых сцен.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Реализация игры

«6 Minutes» – это компьютерная игра с противником нейросетью. Игра представляет собой «top-down» шутер.

Игра состоит из набора ассетов, таких как скрипты, сцены и объекты. Скрипты управляют логикой объектов на сценах. В Unity встроен физический движок. Он включает законы, правила взаимодействия элементов сцены между собой.

Компьютерная игра «6 Minutes» сделана в формате 2D. Это позволило: облегчить разработку, рисование текстур и увеличить оптимизацию. Двухмерная графика работает с изображением, сформированным в двух измерениях – высоте и ширине. Объектом трехмерной графики является изображение, сформированное в трех измерениях: ширине, высоте и глубине. 2D-игровое пространство состоит из единственного слоя, где персонаж движется и взаимодействует с предметами.

3.1.1 Создание проекта Unity

Начнем разработку игры с создания проекта. Для спроектированной игры необходимо выбрать 2D шаблон. Зададим имя проекту и нажмем на кнопку «Create project». Окно создания нового проекта представлено на рисунке 3.1

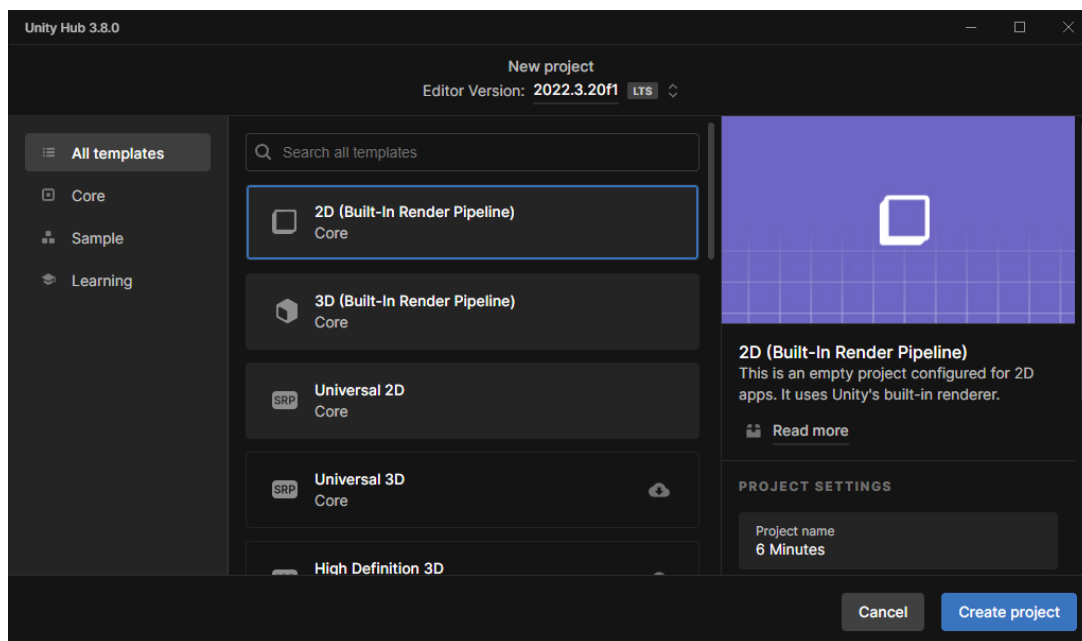


Рисунок 3.1 – Создание нового проекта в Unity Hub

3.1.2 Настройка ML-Agents

ML-Agents (Machine Learning Agents) – это инструмент и набор библиотек, разработанных Unity Technologies, предназначенных для интеграции методов машинного обучения в проекты, создаваемые с использованием Unity. Этот инструмент позволяет разработчикам обучать и использовать интеллектуальные агенты для создания сложных и реалистичных игровых поведений и симуляций.

Приступим к установке ML-Agents. Для корректной работы этого инструмента необходимо установить Python. Его можно легко получить в магазине Microsoft Store (рисунок 3.2)

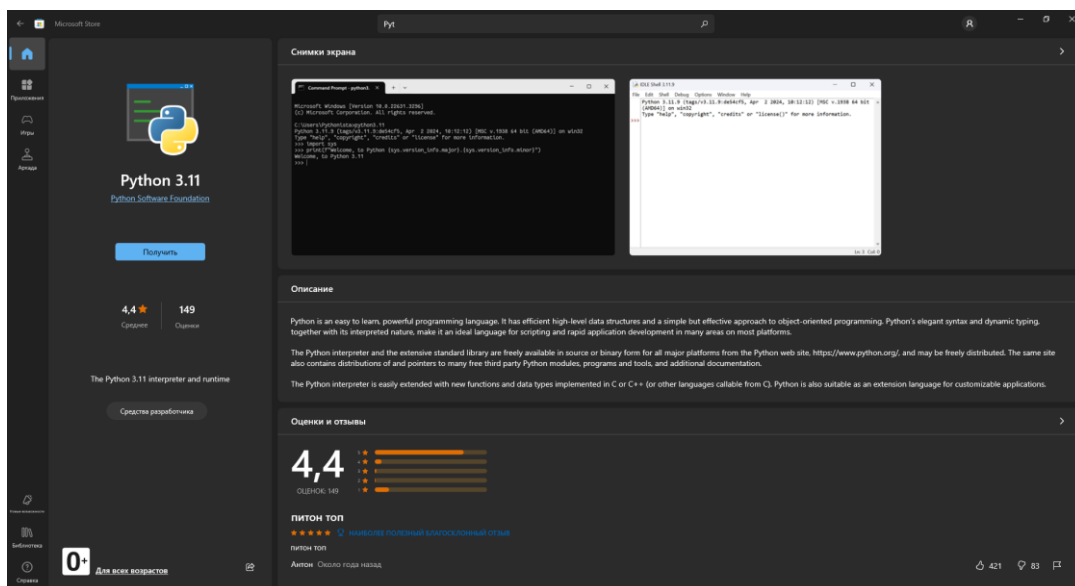


Рисунок 3.2 – Python в Microsoft store

После того как Python установился, необходимо проверить его работоспособность. Сделать это можно с помощью команды «ру». Зайдем в каталог с созданным Unity проектом и проверим версию Python. Процесс проверки изображен на рисунке 3.3

```
C:\Script\Unity Projects\6 Minutes>py
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ^Z
```

Рисунок 3.3 – Проверка работоспособности Python

Когда мы убедились в том, что Python установлен корректно, создадим виртуальное окружение чтобы модули Python не конфликтовали между собой, и запустим его (рисунок 3.4)

```
C:\Script\Unity Projects\6 Minutes>python -m venv venv
C:\Script\Unity Projects\6 Minutes>venv\scripts\activate
(venv) C:\Script\Unity Projects\6 Minutes>
```

Рисунок 3.4 – Работа с виртуальным окружением venv

Далее необходимо установить нужные пакеты python с помощью команды pip install. Список необходимых модулей:

- 1) Mlagents
- 2) Torch, torchvision, torchaudio
- 3) Protobuf
- 4) Packaging
- 5) Onnx

Когда все необходимые пакеты будут установлены, можно начать разработку в Unity

Для начала нужно создать арену, в которой мы запустим тренировку нейросети. Создадим простую прямоугольную арену и разместим на ней игрока и противника.



Рисунок 3.5 – Созданная арена

Обучение нейросети в Unity ML-Agents происходит через взаимодействие между игровым движком Unity и Python-библиотеками, обеспечивая среду для обучения агентов на основе методов машинного обучения. Объект, которому предстоит пройти обучение, называется агентом. Перед тем как начать обучение, необходимо написать скрипт для агента. В данном случае агентом будет противник, так как игроком будет управлять человек.

Обучение агента происходит на основе трех важных аспектах ML-Agent: Наблюдения, действия и вознаграждения.

Наблюдения: Агенты собирают данные об окружающей среде через сенсоры. Это могут быть данные о положении объектов, расстояниях до препятствий, скорости и т.д.

Действия: Агенты выполняют действия на основе полученных наблюдений. Действия могут быть непрерывными (например, управление скоростью) или дискретными (например, выбор направления движения).

Вознаграждения: Агенты получают вознаграждения за выполнение определенных действий или достижение целей. Вознаграждения могут быть положительными (например, за успешное выполнение задачи) или отрицательными (например, за столкновение с препятствием).

Начнем с наблюдения

```
public override void CollectObservations(VectorSensor sensor)
{
    // Добавляем наблюдение за текущим местоположением объекта
    sensor.AddObservation((Vector2)transform.localPosition);

    // Проверяем, задан ли target перед добавлением наблюдения за его местоположением
    if (target != null)
    {
        sensor.AddObservation((Vector2)target.localPosition);
    }
    else
    {
        sensor.AddObservation(Vector2.zero);
    }
}
```

Листинг 3.1 – Метод наблюдения

Метод `CollectObservations` в `Unity ML-Agents` выполняет сбор данных (наблюдений) об окружающей среде, которые используются агентом для принятия решений. Наблюдения могут включать любую информацию, которая может быть полезна для агента, например, положение объектов, состояние самого агента, расстояния до препятствий и т.д. Эти данные передаются в нейронную сеть, которая затем определяет, какие действия агент должен выполнить.

В данном случае, метод получает информацию о своей позиции, и о позиции игрока.

```

public override void OnActionReceived(ActionBuffers actions)
{
    float moveX = actions.ContinuousActions[0];
    float moveY = actions.ContinuousActions[1];

    transform.localPosition += new Vector3(moveX, moveY) * Time.deltaTime * movementSpeed;

    AddReward(-0.01f);
}

```

Листинг 3.2 – Метод действий

Метод `OnActionReceived` в Unity ML-Agents выполняет действия, которые агент выбирает на основе вывода нейронной сети. Этот метод принимает входные данные (действия) от модели машинного обучения и применяет их к агенту в игровом мире. Основная цель метода – обработка действий, выполнение логики игры, вычисление вознаграждений и, при необходимости, завершение эпизода.

В данном случае метод просто двигает объект по арене. Однако каждое действие метод отнимает 0.01 от награды сети, что должно способствовать быстрому движению объекта к цели.

```

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.TryGetComponent(out Goal target))
    {
        AddReward(10f);
        EndEpisode();
    }
    else if (collision.TryGetComponent(out Wall wall))
    {
        AddReward(-2f);
        EndEpisode();
    }
}

```

Листинг 3.3 – Метод награждения

Награды служат для направления процесса обучения, поощряя или наказывая агента за его действия, что помогает ему лучше понимать, какие действия приводят к желаемым результатам. Это основная концепция в методах обучения с подкреплением.

В данном случае, при достижении цели, к награде добавляется 10. При соприкосновении со стеной, награда уменьшается на 2.

Теперь, когда мы написали скрипт для начала обучения сети, добавим его в Inspector объекта Enemy. При добавлении скрипта на объект врага, к нему автоматически добавится скрипт «Behavior Parameters» (Рисунок 3.6)

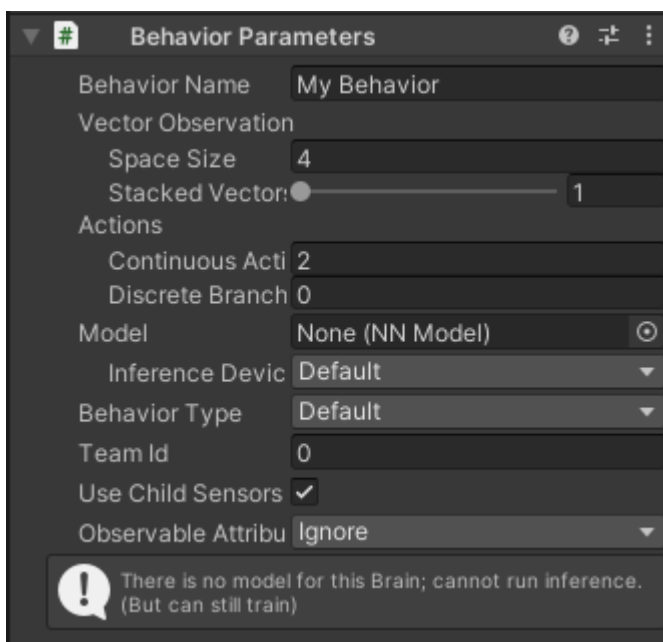


Рисунок 3.6 – Компонент «Behavior Parameters»

Из важного можно выделить следующие параметры:

Space Size. Этот параметр определяет, какие данные агент воспринимает из окружающей среды и какие действия он может предпринимать.

Continuous Actions. Непрерывные действия – это концепция, где агент может выбирать действия из непрерывного диапазона значений. Это означает, что в каждый момент времени агент может выбирать любое значение из определенного интервала, а не ограничиваться дискретным набором конкретных действий.

Примерами непрерывных действий могут быть ускорение, скорость, угол поворота и т. д., где агент может выбирать любое значение в определенном диапазоне. Непрерывные действия часто используются в задачах, где агенту необходимо осуществлять плавное управление или имеется большое пространство действий.

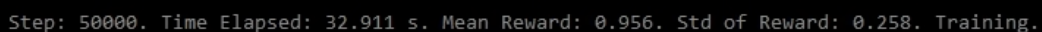
Model. Термин Модель обычно относится к нейронной сети, которая обучается агенту для выполнения конкретной задачи в среде. Модель

принимает входные данные (наблюдения) от среды и выдает соответствующие действия, которые агент должен предпринять в ответ на эти данные. После обучения нейросети, мы будем использовать ее модель в объектах противников.

Добавляем объекту «Enemy» компонент «Decision Requester» и приступаем к обучению нейросети.

3.1.3 Обучение агента

Процесс обучения запустить очень просто. Для этого достаточно написать в командной строке команду «mlagents-learn». При повторном запуске обучения необходимо будет указать идентификатор запуска. После ввода команды, необходимо запустить проект в Unity. Если все было сделано правильно, нейросеть начнет хаотично двигаться. Если агент столкнется со стеной или с игроком, эпизод начнется заново. В командной строке выводится краткая статистика этапов обучения (рисунок 3.7)



```
Step: 50000. Time Elapsed: 32.911 s. Mean Reward: 0.956. Std of Reward: 0.258. Training.
```

Рисунок 3.7 – Статистика обучения в командной строке

Все результаты обучения будут храниться в папке «results». На рисунке 3.8 Представлен пример содержимого каталога результата обучения

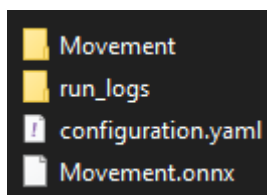


Рисунок 3.8 – Содержимое каталога обучения

Спустя несколько часов обучения у нас есть готовая модель, которая осуществляет движение противников за игроком. Эта модель имеет формат .onnx. Гиперпараметры этой модели можно посмотреть в файле configuration.yaml

```

hyperparameters:
  batch_size: 1024
  buffer_size: 10240
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambda: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
  beta_schedule: linear
  epsilon_schedule: linear

```

Листинг 3.4 – Гиперпараметры модели

Чтобы отменить обучение при следующих запусках программы, необходимо выбрать пункт «Inference Only» для параметра «Behavior Type». Осталось только загрузить модель нейросети в объект противника и создать из него префаб. Загрузить модель в объект, можно перетаскив файл Movement.onnx в поле параметра «Model». Итоговые настройки компонента «Behavior Parameters» представлены на рисунке 3.9

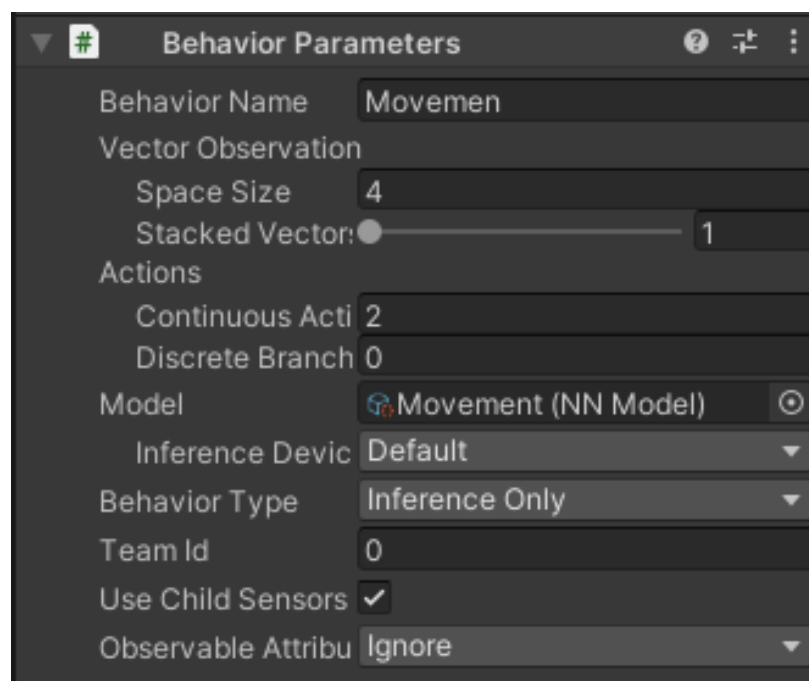


Рисунок 3.9 – Компонент Behavior Parameters

3.1.4 Скрипты, функции, ассеты

Скрипт «Player» отвечает за взаимодействие персонажа с игрой. Основными действиями для этого скрипта является движение и атака. Данный скрипт имеет следующие функции:

Функция `FixedUpdate`. Она вызывается на равных интервалах, определенных физическим движком. В данной функции игра получает ввод игрока и двигает его в зависимости от ввода. Также в этой функции осуществляется механика стрельбы и реализуется повышение сложности со временем.

```
void FixedUpdate()
{
    // Модификатор сложности
    projectileSpeed += Time.deltaTime / 100;
    fireRate -= Time.deltaTime / 2000;
    moveSpeed += Time.deltaTime / 100;

    // Получаем ввод от игрока для движения
    float horizontalInput = Input.GetAxisRaw("Horizontal");
    float verticalInput = Input.GetAxisRaw("Vertical");

    // Передвигаем треугольник в зависимости от ввода
    Vector3 moveDirection = new Vector3(horizontalInput, verticalInput, 0f).normalized;
    transform.position += moveDirection * moveSpeed * Time.deltaTime;

    LAMouse();

    // Проверяем, нажата ли кнопка для стрельбы (например, пробел)
    if (Input.GetKey(KeyCode.Mouse0) && Time.time >= nextFireTime)
    {
        // Вызываем функцию для стрельбы
        Shoot();

        // Устанавливаем время для следующего выстрела
        nextFireTime = Time.time + fireRate;
    }
}
```

Листинг 3.5 – Функция `FixedUpdate` скрипта `Player`

Функция `LAMouse`. Эта функция обеспечивает поворот игрока в направлении мыши.

```
private void LAMouse()
{
    Vector2 direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - m_transform.position;
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle - 90, Vector3.forward);
    m_transform.rotation = rotation;
}
```

Листинг 3.6 – Функция `LAMouse` скрипта `Player`

Функция Shoot. Данная функция обеспечивает запуск снарядов, в направлении взгляда игрока.

```
void Shoot()
{
    // Создаем экземпляр снаряда в позиции firePoint
    GameObject projectile = Instantiate(projectilePrefab, firePoint.position, firePoint.rotation);

    // Получаем компонент Rigidbody2D снаряда
    Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();

    // Придаем снаряду скорость движения вперед
    rb.velocity = firePoint.up * projectileSpeed;
}
```

Листинг 3.7 – Функция Shoot скрипта Player

Функция Die. Она вызывается, когда у игрока закончились жизни. Завершает игру и загружает главное меню

```
private void Die()
{
    // Логика смерти игрока
    SceneManager.LoadScene("MainMenu");
}
```

Листинг 3.8 – Функция Die скрипта Player

Функция PlayerDamage. Данная функция используется для получения урона игроком при столкновении с противником. После получения урона, у игрока будет несколько кадров неуязвимости. С каждым новым столкновением с врагов, у игрока исчезают жизни.

```

public void PlayerDamage()
{
    GameObject life1 = GameObject.Find("Life1");
    GameObject life2 = GameObject.Find("Life2");
    GameObject life3 = GameObject.Find("Life3");

    Renderer lifeRenderer1 = life1.GetComponent<Renderer>();
    Renderer lifeRenderer2 = life2.GetComponent<Renderer>();
    Renderer lifeRenderer3 = life3.GetComponent<Renderer>();

    if (isInvulnerable)
    {
        return;
    }

    playerHealthPoints -= 1;
    cameraF.TriggerShake();
    if (playerHealthPoints == 2)
    {
        lifeRenderer1.enabled = false;
    }
    else if (playerHealthPoints == 1)
    {
        lifeRenderer2.enabled = false;
    }
    if (playerHealthPoints <= 0)
    {
        lifeRenderer3.enabled = false;
        Die();
    }
    else
    {
        StartCoroutine(InvulnerabilityCoroutine());
    }
}

```

Листинг 3.9 – Функция PlayerDamage скрипта Player

Переменные скрипта:

- 1) moveSpeed. Скорость движения игрока
- 2) projectileSpeed. Скорость полета снаряда
- 3) NextFireTime. Перерыв между стрельбой
- 4) fireRate. Интервал между снарядами
- 5) PlayerHealthPoints. Количество жизней у игрока
- 6) invulnerabilityDuration. Время, в течении которого игрок неуязвим после получения урона
- 7) isInvulnerable. Переменная, которая показывает неуязвим ли игрок в данный момент

Скрипт «ammo» отвечает за логику запускаемых игроком снарядов. Имеет один метод регистрации попадания (листинг 3.10). Если цель попадания имеет тег «Enemy», то у нее вызывается функция получения урона и генерируется эффект попадания (рисунок 3.10), после чего снаряд уничтожается. Если цель имеет тег «Wall», то снаряд уничтожается без последствий.

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Enemy"))
    {
        EnemyHealth enemy = other.GetComponent<EnemyHealth>();

        if (enemy != null)
        {
            enemy.TakeDamage();
            Instantiate(hitEffect, transform.position, Quaternion.identity);
        }

        Destroy(gameObject);
    }
    else if (other.CompareTag("Wall"))
    {
        Destroy(gameObject);
    }
}
```

Листинг 3.10 – Функция OnTriggerEnter2D скрипта Ammo



Рисунок 3.10 – Эффект попадания

Скрипт «CameraFollow» предназначен для красивого движения камеры за игроком. Он имеет две функции: LateUpdate и TriggerShake

Функция LateUpdate в Unity вызывается после выполнения всех методов Update в каждом кадре. Он обычно используется для выполнения действий, которые должны происходить после обновления всех объектов в кадре. В данном случае он управляет движением камеры и вызывает тряску экрана (листинг 3.11).

```
void LateUpdate()
{
    if (shakeDuration > 0)
    {
        transform.localPosition = transform.position + Random.insideUnitSphere * shakeMagnitude;
        shakeDuration -= Time.deltaTime * dampingSpeed;
    }
    else
    {
        shakeDuration = 0f;
        Vector3 desiredPosition = player.position + offset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        transform.position = smoothedPosition;
    }
}
```

Листинг 3.11 – Функция LateUpdate скрипта CameraFollow

Функция TriggerShake устанавливает время тряски экрана на назначенную. При этом функция LateUpdate проигрывает эту тряску. Эта функция представлена на листинге 3.12

```
public void TriggerShake()
{
    shakeDuration = initialDuration;
}
```

Листинг 3.12 Функция TriggerShake скрипта CameraFollow

Скрипт DeleteEffect предназначен для удаления проигранных эффектов для повышения оптимизации. Имеет единственную функцию Start, которая при запуске скрипта уничтожает ее объект (листинг 3.13).

```
private void Start()
{
    // Уничтожаем объект после проигрывания анимации
    Destroy(gameObject, duration);
}
```

Листинг 3.13 – Функция Start скрипта DeleteEffect

Скрипт EnemyHealth управляет временем жизни врагов. Имеет две функции: TakeDamage и OnTriggerEnter2D

Вызов функции TakeDamage уменьшает здоровье врага на 1, а при достижении переменной health значения 0, уничтожает объект врага (листинг 3.14).

```
public void TakeDamage()
{
    health--;

    if (health <= 0)
    {
        Instantiate(DeathEffect, transform.position, Quaternion.identity);
        Destroy(gameObject);
    }
}
```

Листинг 3.14 – Функция TakeDamage скрипта EnemyHealth

Функция OnTriggerEnter2D необходима для регистрации столкновения противника с игроком. При вызове этой функции, вызывается функция PlayerDamage скрипта Player (листинг 3.15).

```
void OnTriggerEnter2D(Collider2D other)
{
    Player player = other.GetComponent<Player>();

    if (other.CompareTag("Player"))
    {
        player.PlayerDamage();
    }
}
```

Листинг 3.15 – Функция OnTriggerEnter2D скрипта EnemyHealth

Скрипт EnemySpawn отвечает за случайное появление врагов на арене. Он имеет 3 функции: Update, SpawnEnemy, GetRandomEdgePosition.

Функция Update (листинг 3.16) вызывается каждый кадр и вызывает функцию SpawnEnemy (листинг 3.17), которая в свою очередь генерирует врагов на краях арены. С повышением сложности, интервал между появлением врагов уменьшается.

```

void Update()
{
    spawnInterval -= Time.deltaTime / 400;

    spawnTimer += Time.deltaTime; // Увеличиваем таймер

    // Если прошло время для спауна нового врага
    if (spawnTimer >= spawnInterval)
    {
        SpawnEnemy(); // Спауним врага
        spawnTimer = 0f; // Сбрасываем таймер
    }
}

```

Листинг 3.16 – Функция Update скрипта EnemySpawn

```

void SpawnEnemy()
{
    Vector3 spawnPosition = GetRandomEdgePosition();
    GameObject agent = Instantiate(enemyPrefab, spawnPosition, Quaternion.identity); // Создаем врага из префаба
    TestAI agentScript = agent.GetComponent<TestAI>();
    if (agentScript != null)
    {
        agentScript.target = target;
    }
}

```

Листинг 3.17 – Функция SpawnEnemy скрипта EnemySpawn

Функция GetRandomEdgePosition вычисляет случайную точку на крае арены. Ее код представлен в листинге 3.18

```

Vector3 GetRandomEdgePosition()
{
    int edge = Random.Range(0, 4); // 0 - верх, 1 - низ, 2 - лево, 3 - право
    Vector3 spawnPosition = Vector3.zero;

    switch (edge)
    {
        case 0: // Верхний край
            spawnPosition = new Vector3(Random.Range(topLeft.x, topRight.x), topLeft.y, 0f);
            break;
        case 1: // Нижний край
            spawnPosition = new Vector3(Random.Range(bottomLeft.x, bottomRight.x), bottomLeft.y, 0f);
            break;
        case 2: // Левый край
            spawnPosition = new Vector3(topLeft.x, Random.Range(bottomLeft.y, topLeft.y), 0f);
            break;
        case 3: // Правый край
            spawnPosition = new Vector3(topRight.x, Random.Range(bottomRight.y, topRight.y), 0f);
            break;
    }

    return spawnPosition;
}

```

Листинг 3.18 – Функция GetRandomEdgePosition скрипта EnemySpawn

Скрипт Hover осуществляет изменение размера текста при наведении мыши. Он имеет 3 функции: Start, OnPointerEnter, OnPointerExit.

Функция Start обеспечивает инициализацию компонента TextMeshPro. Ее код представлен в листинге 3.19

```
void Start()
{
    if (buttonText == null)
    {
        buttonText = GetComponentInChildren<TextMeshProUGUI>();
    }

    if (buttonText != null)
    {
        originalFontSize = buttonText.fontSize;
    }
    else
    {
        Debug.LogError("TextMeshProUGUI component not found on button.");
    }
}
```

Листинг 3.19 – Функция Start скрипта Hover

Функции OnPointerEnter и OnPointerExit меняют непосредственно размер шрифта у кнопок. Их код представлен в листинге 3.20

```
Ссылка: 2
public void OnPointerEnter(PointerEventData eventData)
{
    if (buttonText != null)
    {
        buttonText.fontSize = hoverFontSize;
    }
}

Ссылка: 2
public void OnPointerExit(PointerEventData eventData)
{
    if (buttonText != null)
    {
        buttonText.fontSize = originalFontSize;
    }
}
```

Листинг 3.20 – Функции OnPointerEnter и OnPointerExit скрипта Hover

Скрипт MenuButtons (листинг 3.21) имеет 2 функции для работы со сценами. Функция SceneSwitch загружает игровую сцену при нажатии на кнопку. Функция QuitGame закрывает приложение при нажатии на кнопку.

```

Ссылка: 0
public void SceneSwitch()
{
    SceneManager.LoadScene("GameScene");
}

Ссылка: 0
public void QuitGame()
{
    Application.Quit();
}

```

Листинг 3.21 – Функции SceneSwitch и QuitGame скрипта MenuButtons

Скрипт TimerForDifficulty отвечает за игровой таймер. Имеет 2 функции: Update и endGame.

Функция Update (листинг 3.22) обновляет таймер, а функция endGame (листинг 3.23), закружает главное меню. endGame вызывается сразу после того как игрок победил

```

void Update()
{
    currentTime -= Time.deltaTime;

    if (currentTime < 0)
    {
        currentTime = 0;
        endGame();
    }

    int minutes = Mathf.FloorToInt(currentTime / 60F);
    int seconds = Mathf.FloorToInt(currentTime % 60F);
    timerText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
}

```

Листинг 3.22 – Функция Update скрипта TimerForDifficulty

```

Ссылка: 0
void endGame()
{
    SceneManager.LoadScene("MainMenu");
}

```

Листинг 3.23 – Функция endGame скрипта TimerForDifficulty

3.2 Тестирование

Для приложения с противником на основе нейросети было проведено успешное тестирование. В процессе тестирования не выявлено никаких дефектов, что свидетельствует о стабильной работе игры и ее соответствии требованиям. Тестирование включало в себя проверку всех основных функций игры, взаимодействие с противником на нейросети, а также адекватность и эффективность поведения противника в различных сценариях игры. Результаты тестирования подтверждают успешную реализацию и работоспособность игры, обеспечивая высокий уровень качества и удовлетворение потребностей пользователей.

3.3 Руководство пользователя

3.3.1 Введение

Приложение «6 Minutes» предназначено для пользователей, желающих испытать свои навыки в игре против противника, управляемого нейросетью. Оно может быть использовано как любителями игр, так и разработчиками для изучения поведения искусственного интеллекта в игровой среде.

Приложение рассчитано на пользователей с базовыми навыками работы с компьютерами и игровыми приложениями. Не требуется специального технического образования или знаний в области программирования. Пользователь должен уметь запускать приложение на своем устройстве, ориентироваться в интерфейсе и выполнять основные игровые действия.

Приложение «6 Minutes» предназначено для:

Обеспечения интересного и увлекательного игрового опыта с использованием противника, управляемого нейросетью.

Демонстрации возможностей и поведения нейросетевого противника в различных игровых сценариях.

Предоставления игрокам возможности изучить стратегии и тактики для успешного противостояния ИИ-противнику.

Для использования приложения «6 Minutes» необходим компьютер с операционной системой Windows, достаточное свободное место на устройстве для установки приложения и хранения игровых данных.

3.3.2 Описание операций

Для установки приложения необходимо скачать каталог с проектом из репозитория в GitHub (<https://github.com/StopITnowFAST/6-minutes>). Далее необходимо найти файл «Neural Game.exe» и запустить его. Появится главное меню приложения. Чтобы начать игру, нужно нажать на кнопку «Play»

Приложение «Neuro-Game» разработано для создания увлекательного игрового процесса и демонстрации возможностей искусственного интеллекта в играх. Данное руководство пользователя предоставляет полную информацию для эффективного использования всех возможностей игры с противником на нейросети.

Выводы

В процессе разработки были решены множество технических задач, связанных с интеграцией нейронной сети в игровой процесс, обеспечением стабильной работы приложения и созданием увлекательного игрового процесса. Разработка игры с нейросетевым противником является увлекательным и интересным процессом, который обогащает знания и опыт в области разработки игр и искусственного интеллекта.

ЗАКЛЮЧЕНИЕ

Разработка игры с противником, управляемым нейросетью, представляла собой захватывающее и инновационное путешествие в мир современных технологий и искусственного интеллекта. Цель заключалась в создании уникального игрового опыта, который бы предложил пользователям интересное взаимодействие с нейросетевым противником и вызвал у них интерес к изучению поведения искусственного интеллекта в игровой среде.

В ходе работы над проектом были успешно реализованы ключевые аспекты:

Проведено обширное исследование в области искусственного интеллекта и игровой разработки для понимания основных принципов и методов создания игры с нейросетевым противником.

Был разработан детальный план игры, определены основные механики, интерфейс и структура игровых сцен для обеспечения увлекательного игрового процесса.

Успешно интегрирован нейросетевой противник в игровую среду на основе Unity3D и ML-Agents.

Была создана захватывающая игра, позволяющая игрокам насладиться увлекательным игровым процессом.

СПИСОК ЛИТЕРАТУРЫ

1. Галенкин С. Маркетинг игр [Текст] / С. Галенкин. – Санкт-Петербург: Питер, 2014 – 78 с.
2. Карпекин И.К. Создаем 2D-игру на Unity: инструкция для новичка / Карпекин И.К. [Электронный ресурс] // ProgLib : [сайт]. – URL: <https://proglib.io/p/sozdaem-2d-igru-na-unity-instrukciya-dlya-novichka-2020-09-01> (дата обращения: 31.05.2024).
3. Владимир С. Что такое Unity и почему он так популярен / Владимир С. [Электронный ресурс] // Netology : [сайт]. – URL: <https://netology.ru/blog/01-2022-unity-development> (дата обращения: 31.05.2024).
4. Роль компьютерных игр в жизни современного общества / [Электронный ресурс] // Почемуха : [сайт]. – URL: <http://pochemuha.ru/rol-kompyuternyx-igr-v-zhizni-sovremennogo-obshhestva> (дата обращения: 31.05.2024).
5. История компьютерных игр / [Электронный ресурс] // Wikipedia : [сайт]. – URL: https://ru.wikipedia.org/wiki/История_компьютерных_игр (дата обращения: 31.05.2024).
6. Игровая индустрия: геймдев (gamedev) / [Электронный ресурс] // HSBI : [сайт]. – URL: <https://hsbi.hse.ru/articles/igrovaya-industriya-geymdev/> (дата обращения: 31.05.2024).
7. Знакомство с C#: интерактивные руководства | Microsoft Learn / [Электронный ресурс] // learn.microsoft : [сайт]. – URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/tutorials/> (дата обращения: 31.05.2024).
8. Платформа Unity для разработки в реальном времени | Движок для 3D, 2D, VR и AR / [Электронный ресурс] // unity : [сайт]. – URL: <https://unity.com/ru> (дата обращения: 31.05.2024).

Листинг А1 – Скрипт Player

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;
using UnityEngine.UIElements;
using UnityEngine.SceneManagement;

public class Player : MonoBehaviour
{
    // Движение игрока
    public float moveSpeed = 5f;

    // Огневые параметры
    private float projectileSpeed = 10f;
    private float nextFireTime = 0f;
    private float fireRate = 0.2f;

    // Жизненные параметры
    private int playerHealthPoints = 3;
    private float invulnerabilityDuration = 2f;
    private bool isInvulnerable = false;

    // Объекты
    private CameraFollow cameraF;
    private Transform m_transform;
    public GameObject projectilePrefab;
    public Transform firePoint;

    private void Start()
    {
        m_transform = this.transform;
        cameraF = Camera.main.GetComponent<CameraFollow>();
    }

    public void PlayerDamage()
    {
        GameObject life1 = GameObject.Find("Life1");
        GameObject life2 = GameObject.Find("Life2");
        GameObject life3 = GameObject.Find("Life3");

        Renderer lifeRenderer1 = life1.GetComponent<Renderer>();
        Renderer lifeRenderer2 = life2.GetComponent<Renderer>();
        Renderer lifeRenderer3 = life3.GetComponent<Renderer>();

        if (isInvulnerable)
```

```

        {
            return;
        }

        playerHealthPoints -= 1;
        cameraF.TriggerShake();
        if (playerHealthPoints == 2)
        {
            lifeRenderer1.enabled = false;
        }
        else if (playerHealthPoints == 1)
        {
            lifeRenderer2.enabled = false;
        }
        if (playerHealthPoints <= 0)
        {
            lifeRenderer3.enabled = false;
            Die();
        }
        else
        {
            StartCoroutine(InvulnerabilityCoroutine());
        }
    }

    private void Die()
    {
        // Логика смерти игрока
        SceneManager.LoadScene("MainMenu");
    }

    private IEnumerator InvulnerabilityCoroutine()
    {
        isInvulnerable = true;
        yield return new
        WaitForSeconds(invulnerabilityDuration);
        isInvulnerable = false;
    }

    private void LAMouse()
    {
        Vector2 direction =
        Camera.main.ScreenToWorldPoint(Input.mousePosition) -
        m_transform.position;
        float angle = Mathf.Atan2(direction.y, direction.x) *
        Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(angle - 90,
        Vector3.forward);
        m_transform.rotation = rotation;
    }

    void FixedUpdate()
    {

```

```

// Модификатор сложности
projectileSpeed += Time.deltaTime / 100;
fireRate -= Time.deltaTime / 2000;
moveSpeed += Time.deltaTime / 100;

// Получаем ввод от игрока для движения
float horizontalInput = Input.GetAxisRaw("Horizontal");
float verticalInput = Input.GetAxisRaw("Vertical");

// Передвигаем треугольник в зависимости от ввода
Vector3 moveDirection = new Vector3(horizontalInput,
verticalInput, 0f).normalized;
transform.position += moveDirection * moveSpeed *
Time.deltaTime;

LAMouse();

// Проверяем, нажата ли кнопка для стрельбы (например,
пробел)
if (Input.GetKey(KeyCode.Mouse0) && Time.time >=
nextFireTime)
{
    // Вызываем функцию для стрельбы
    Shoot();

    // Устанавливаем время для следующего выстрела
    nextFireTime = Time.time + fireRate;
}

void Shoot()
{
    // Создаем экземпляр снаряда в позиции firePoint
    GameObject projectile = Instantiate(projectilePrefab,
firePoint.position, firePoint.rotation);

    // Получаем компонент Rigidbody2D снаряда
    Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();

    // Придаем снаряду скорость движения вперед
    rb.velocity = firePoint.up * projectileSpeed;
}
}

```

Листинг А2 – Скрипт Ammo

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

public class Ammo : MonoBehaviour
{
    public GameObject hitEffect;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Enemy"))
        {
            EnemyHealth enemy =
other.GetComponent<EnemyHealth>();

            if (enemy != null)
            {
                enemy.TakeDamage();
                Instantiate(hitEffect, transform.position,
Quaternion.identity);
            }

            Destroy(gameObject);
        }
        else if (other.CompareTag("Wall"))
        {
            Destroy(gameObject);
        }
    }
}
```


Листинг А3 – Скрипт CameraFollow

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

public class CameraFollow : MonoBehaviour
{
    public Transform player;
    public Vector3 offset;
    public float smoothSpeed = 0.125f;

    public float shakeDuration = 0.2f;
    public float shakeMagnitude = 0.5f;
    public float dampingSpeed = 1.0f;

    private float initialDuration;

    void Start()
    {
        initialDuration = shakeDuration;

        if (offset == Vector3.zero)
        {
            offset = new Vector3(0, 0, -10);
        }
    }

    void LateUpdate()
    {
        if (shakeDuration > 0)
        {
            transform.localPosition = transform.position +
Random.insideUnitSphere * shakeMagnitude;

            shakeDuration -= Time.deltaTime * dampingSpeed;
        }
        else
        {
            shakeDuration = 0f;
            Vector3 desiredPosition = player.position + offset;
            Vector3 smoothedPosition =
Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
            transform.position = smoothedPosition;
        }
    }

    public void TriggerShake()
    {
        shakeDuration = initialDuration;
    }
}
```

Листинг А4 – Скрипт DeleteEffect

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

public class DeleteEffect : MonoBehaviour
{
    public float duration = 1f; // Длительность эффекта

    private void Start()
    {
        // Уничтожаем объект после проигрывания анимации
        Destroy(gameObject, duration);
    }
}
```

Листинг А5 – Скрипт EnemyHealth

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

public class EnemyHealth : MonoBehaviour
{
    public int health = 3; // Количество попаданий до
    исчезновения
    public GameObject enemyPrefab; // Префаб врага, который
    будет спавниться
    public Vector2 mapSize = new Vector2(23.0f, 65.0f); //
    Размер карты по ширине и высоте
    public GameObject DeathEffect;

    // Метод для обработки попадания
    public void TakeDamage()
    {
        health--;

        if (health <= 0)
        {
            Instantiate(DeathEffect, transform.position,
            Quaternion.identity);
            Destroy(gameObject);
        }
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        Player player = other.GetComponent<Player>();

        if (other.CompareTag("Player"))
        {
            player.PlayerDamage();
        }
    }
}
```

Листинг А6 – Скрипт EnemySpawn

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

public class EnemySpawn : MonoBehaviour
{
    public GameObject enemyPrefab;
    public float spawnInterval = 5f; // Интервал между спаунами
врагов
    private float spawnTimer = 0f;
    public Transform target; // Ссылка на игрока
    public float currentTime;

    // Координаты арены
    private Vector3 topLeft = new Vector3(-33f, 24f, 0f);
    private Vector3 topRight = new Vector3(31f, 24f, 0f);
    private Vector3 bottomLeft = new Vector3(-33f, 2f, 0f);
    private Vector3 bottomRight = new Vector3(31f, 2f, 0f);

    void Update()
    {
        spawnInterval -= Time.deltaTime / 400;

        spawnTimer += Time.deltaTime; // Увеличиваем таймер

        // Если прошло время для спауна нового врага
        if (spawnTimer >= spawnInterval)
        {
            SpawnEnemy(); // Спауним врага
            spawnTimer = 0f; // Сбрасываем таймер
        }
    }

    void SpawnEnemy()
    {
        Vector3 spawnPosition = GetRandomEdgePosition();
        GameObject agent = Instantiate(enemyPrefab,
spawnPosition, Quaternion.identity); // Создаем врага из префаба
        TestAI agentScript = agent.GetComponent<TestAI>();
        if (agentScript != null)
        {
            agentScript.target = target;
        }
    }

    Vector3 GetRandomEdgePosition()
    {
        int edge = Random.Range(0, 4); // 0 - верх, 1 - низ, 2 -
лево, 3 - право
    }
}
```

```

    Vector3 spawnPosition = Vector3.zero;

    switch (edge)
    {
        case 0: // Верхний край
            spawnPosition = new
Vector3(Random.Range(topLeft.x, topRight.x), topLeft.y, 0f);
            break;
        case 1: // Нижний край
            spawnPosition = new
Vector3(Random.Range(bottomLeft.x, bottomRight.x), bottomLeft.y,
0f);
            break;
        case 2: // Левый край
            spawnPosition = new Vector3(topLeft.x,
Random.Range(bottomLeft.y, topLeft.y), 0f);
            break;
        case 3: // Правый край
            spawnPosition = new Vector3(topRight.x,
Random.Range(bottomRight.y, topRight.y), 0f);
            break;
    }

    return spawnPosition;
}
}

```

Листинг А7 – Скрипт Hover

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

public class Hover : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler
{
    public TextMeshProUGUI buttonText;
    public float hoverFontSize = 144f; // Размер шрифта при
наведении
    private float originalFontSize;

    void Start()
    {
        if (buttonText == null)
        {
            buttonText =
GetComponentInChildren<TextMeshProUGUI>();
        }

        if (buttonText != null)
        {
            originalFontSize = buttonText.fontSize;
        }
        else
        {
            Debug.LogError("TextMeshProUGUI component not found
on button.");
        }
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (buttonText != null)
        {
            buttonText.fontSize = hoverFontSize;
        }
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (buttonText != null)
        {
            buttonText.fontSize = originalFontSize;
        }
    }
}
```

Листинг A8 – Скрипт MenuButtons

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

public class MenuButtons : MonoBehaviour
{
    public void SceneSwitch()
    {
        SceneManager.LoadScene("GameScene");
    }

    public void QuitGame()
    {
        Application.Quit();
    }
}
```

Листинг А9 – Скрипт TimerForDifficulty

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

public class TimerForDifficulty : MonoBehaviour
{
    [Header("Component")]
    public TextMeshProUGUI timerText;

    [Header("Timer Settings")]
    private float currentTime = 360f;
    public bool countUp;

    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        currentTime -= Time.deltaTime;

        if (currentTime < 0)
        {
            currentTime = 0;
            endGame();
        }

        int minutes = Mathf.FloorToInt(currentTime / 60F);
        int seconds = Mathf.FloorToInt(currentTime % 60F);
        timerText.text = string.Format("{0:00}:{1:00}", minutes,
seconds);
    }

    void endGame()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```


Листинг A10 – Скрипт TestAI

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using System;
using Unity.MLAgents.Sensors;
using Random = UnityEngine.Random;
using Unity.MLAgents.Policies;
using static UnityEngine.GraphicsBuffer;

public class TestAI : Agent
{
    public Transform target;
    public float movementSpeed = 5f;

    public override void Initialize()
    {
        var behaviorParameters =
GetComponent<BehaviorParameters>();
        behaviorParameters.BehaviorType =
BehaviorType.InferenceOnly;
    }

    public override void CollectObservations(VectorSensor
sensor)
    {
        // Добавляем наблюдение за текущим местоположением
объекта
        sensor.AddObservation((Vector2)transform.localPosition);

        // Проверяем, задан ли target перед добавлением
наблюдения за его местоположением
        if (target != null)
        {
            sensor.AddObservation((Vector2)target.localPosition);
        }
        else
        {
            sensor.AddObservation(Vector2.zero);
        }
    }

    public override void OnActionReceived(ActionBuffers actions)
    {
        float moveX = actions.ContinuousActions[0];
        float moveY = actions.ContinuousActions[1];
    }
}
```

```

        transform.localPosition += new Vector3(moveX, moveY) *
Time.deltaTime * movementSpeed;
        AddReward(-0.01f);
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<float> continuousActions =
actionsOut.ContinuousActions;
        continuousActions[0] = Input.GetAxisRaw("Horizontal");
        continuousActions[1] = Input.GetAxisRaw("Vertical");
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.TryGetComponent(out Goal target))
        {
            AddReward(10f);
            EndEpisode();
        }
        else if (collision.TryGetComponent(out Wall wall))
        {
            AddReward(-2f);
            EndEpisode();
        }
    }
}

```

Листинг Б1 – configuration.yaml

```
default_settings: null
behaviors:
  Movement:
    trainer_type: false
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambda: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 500000
    max_steps: 500000
    time_horizon: 64
    summary_freq: 50000
    threaded: false
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
```

```
num_envs: 1
num_areas: 1
seed: -1
max_lifetime_restarts: 10
restarts_rate_limit_n: 1
restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 20
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: 22_the_best
  initialize_from: null
  load_model: false
  resume: true
  force: false
  train_model: false
  inference: true
  results_dir: results
torch_settings:
  device: null
debug: false
```