

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ**

**МОДЕЛИРОВАНИЯ ПОВЕДЕНИЯ**

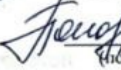
**ПРОТИВНИКОВ В ИГРАХ**

Студент

 30.05.2025  
(подпись, дата)

Суворов Максим Дмитриевич  
(ФИО полностью)

Руководитель

 30.05.2025  
(подпись, дата)

Попова Н.А.  
(фамилия, инициалы)

Нормоконтролёр

 30.05.2025  
(подпись, дата)

Бычков А.С.  
(фамилия, инициалы)

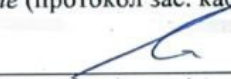
Рецензент

2А. Снегачев  
прекрасно  
(должность, место работы) (подпись, дата)  
40 «КП» Рубин

Васильев А.В.  
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 № 16)

Заведующий кафедрой

  
(подпись)

Митрохин М.А.  
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

5 июня 2025 г. № 1)

Секретарь ГЭК

  
(подпись)

Гурин Е.И.  
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет  
вычислительной техники

Кафедра  
«Вычислительная техника»

«Утверждаю»  
Заведующий кафедрой  
М.А. Митрохин  
«29» сентября 2023г.

**ЗАДАНИЕ  
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Суворов Максим Дмитриевич гр.23ВВИМ1 факультета ВТ

2. Тема работы «Применение нейронных сетей для моделирования поведения противников в играх»

Тема утверждена приказом ПГУ №986/о от 29.09.2023

3. Руководитель магистерской диссертации к.т.н, доцент кафедры «МО и ПЭВМ»  
Попова Н.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Проектирование и разработка системы для обучения нейронных сетей, предназначенных для моделирования поведения противников в играх. Система должна обеспечивать возможность обучения нейронных сетей, сохранения результатов их обучения, корректировки гиперпараметров, а также отображения статистики обучения. Необходимо реализовать поддержку обучения нейронных сетей на графических процессорах. Требуется разработать прототип проекта для демонстрации результатов и работоспособности системы.

### 5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить способы реализации игрового искусственного интеллекта, рассмотреть основные средства создания игр. Необходимо разработать архитектуру проекта и методы его тестирования. Также следует изучить вспомогательные технологии, используемые для разработки, такие как Unity, Python и TensorFlow.

### 6. Календарный график выполнения работы

| № п/п | Наименование этапов работы                                    | Объем работы | Срок выполнения       | Подпись руководителя |
|-------|---------------------------------------------------------------|--------------|-----------------------|----------------------|
| 1     | Изучение предметной области                                   | 11%          | 29.11.2023-29.01.2024 | Гомофф               |
| 2     | Анализ требований                                             | 11%          | 29.01.2024-30.03.2024 | Гомофф               |
| 3     | Выбор технологий, применяемых при проектировании и разработке | 22%          | 30.03.2024-30.05.2024 | Гомофф               |
| 4     | Проектирование и реализация системы                           | 27%          | 01.06.2024-01.09.2024 | Гомофф               |
| 5     | Обучение прототипа нейронной сети                             | 17%          | 02.09.2024-02.10.2024 | Гомофф               |
| 6     | Тестирование и отладка                                        | 6%           | 01.04.2025-29.04.2025 | Гомофф               |
| 7     | Оформление пояснительной записки                              | 6%           | 30.04.2025-25.05.2025 | Гомофф               |

Дата выдачи задания 29.09.2023

Руководитель магистерской диссертации

Гомофф 29.09.2023  
(подпись, дата)

Задание к исполнению принял студент

Сид 29.09.2023  
(подпись, дата)

Работу к защите допустить  
Декан факультета

Сид 02.06.2025  
(подпись, дата)

## РЕФЕРАТ

Пояснительная записка содержит 85 листов, 18 рисунков, 11 таблиц, 23 источника и 1 приложение.

**ИГРОВОЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, МЕТОДЫ ПРОЕКТИРОВАНИЯ, НЕЙРОННЫЕ СЕТИ, МАШИННОЕ ОБУЧЕНИЕ, UNITY**

Целью проекта является разработка программного обеспечения, позволяющего применять нейронные сети для моделирования поведения противников в играх.

Объектом исследования является процесс обучения и моделирования поведения противников в играх.

Предметом исследования является программное обеспечение для моделирования поведения противников в игровых приложениях.

Технологии разработки: Unity, TensorFlow, Python.

Результат работы: спроектированное и разработанное приложение, позволяющее обучать нейронные сети для моделирования поведения противников и прототип проекта демонстрирующий функционал системы.

|           |      |               |                                                                                     |          |                                                                                                           |                              |      |        |
|-----------|------|---------------|-------------------------------------------------------------------------------------|----------|-----------------------------------------------------------------------------------------------------------|------------------------------|------|--------|
|           |      |               |                                                                                     |          | ПГУ 1.09.04.01.14.001 ПЗ                                                                                  |                              |      |        |
| Изм       | Лист | № Докум       | Подпись                                                                             | Дата     | Применение нейронных сетей<br>для моделирования поведения<br>противников в играх<br>Пояснительная записка | Лит.                         | Лист | Листов |
| Разраб.   |      | Суворов М.Д.  |  | 30.05.25 |                                                                                                           |                              |      |        |
| Провер.   |      | Попова Н.А.   |  | 30.05.25 |                                                                                                           |                              | 2    | 85     |
| Н. Контр. |      | Бычков А.С.   |  | 30.05.25 |                                                                                                           | ФВТ, каф. ВТ<br>гр. 23ВВВим1 |      |        |
| Утв.      |      | Митрохин М.А. |  | 30.05.25 |                                                                                                           |                              |      |        |

## СОДЕРЖАНИЕ

|                                                                   |    |
|-------------------------------------------------------------------|----|
| ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....                        | 6  |
| ВВЕДЕНИЕ.....                                                     | 8  |
| 1 Анализ предметной области.....                                  | 10 |
| 1.1 Актуальность.....                                             | 10 |
| 1.2 Обзор методов создания искусственного интеллекта в играх..... | 11 |
| 1.3 Прямое кодирование ИИ.....                                    | 12 |
| 1.4 Использование шаблона конечного автомата.....                 | 13 |
| 1.5 Дерево поведения.....                                         | 15 |
| 1.6 GOAP подход.....                                              | 17 |
| 1.7 Нейронные сети и машинное обучение.....                       | 18 |
| 2 Выбор средств разработки.....                                   | 22 |
| 2.1 Unreal Engine и C++.....                                      | 22 |
| 2.2 Unity и C#.....                                               | 23 |
| 2.3 Godot Engine.....                                             | 24 |
| 2.4 Критериальная оценка сред разработки игровых приложений.....  | 25 |
| 2.5 Выбор интегрированной среды разработки.....                   | 26 |
| 3 Постановка задачи на разработку.....                            | 28 |
| 3.1 Анализ существующих решений.....                              | 28 |
| 3.2 Цель и назначение системы.....                                | 33 |
| 3.3 Требования к программе.....                                   | 34 |
| 3.4 Функциональные требования.....                                | 35 |
| 4 Разработка системы.....                                         | 36 |
| 4.1 Разработка расширения для Unity.....                          | 36 |
| 4.2 Анализ доступных ML алгоритмов.....                           | 43 |
| 4.3 Требования к прототипу для тестирования системы.....          | 44 |
| 4.4 Обзор возможностей обучения через расширение.....             | 46 |
| 4.5 Оценка качества обучения агентов.....                         | 49 |

|     |                                        |    |
|-----|----------------------------------------|----|
| 5   | Тестирование системы.....              | 52 |
| 5.1 | Разработка тестовой среды.....         | 52 |
| 5.2 | Обучение агентов в тестовой среде..... | 57 |
| 5.3 | Результаты обучения.....               | 58 |
|     | ЗАКЛЮЧЕНИЕ.....                        | 62 |
|     | ПРИЛОЖЕНИЕ А.....                      | 66 |

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Искусственный интеллект (ИИ) – это область исследований в области компьютерных наук, которая разрабатывает и изучает методы и программное обеспечение, позволяющие машинам воспринимать окружающую среду и использовать обучение и интеллект для выполнения действий, которые максимально увеличивают их шансы на достижение поставленных целей

ИИ-агент – компонент, который производит наблюдения и выполняет действия в среде. Действия агентов определяются решениями, принимаемыми в соответствии с Политикой.

Политика – механизм принятия решений на основе наблюдений, обычно представляющий собой нейросетевую модель.

Tensor Flow – открытая программная библиотека для машинного обучения, разработанная для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации, достигая качества человеческого восприятия.

Open Neural Network Exchange (ONNX, англ. открытый формат обмена нейронных сетей) – открытая библиотека программного обеспечения для построения нейронных сетей глубокого обучения.

Compute Unified Device Architecture (CUDA, англ. Вычислить унифицированную архитектуру устройства ) – программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia.

Reinforcement learning (RL, англ. обучение с подкреплением) – один из способов машинного обучения, в ходе которого испытываемая система (агент) обучается, взаимодействуя с некоторой средой.

Goal-Oriented Action Planning (GOAP, англ. Целенаправленное планирование действий) – является методикой проектирования ИИ, при

которой цепочка поведения агентов выбирается в реальном времени автоматически для удовлетворения цели.

Finite State Machine (FSM, англ. конечный автомат) – математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.

YAML-файл (Yet Another Markup Language, англ. еще один язык разметки) представляет собой структурированный конфигурационный файл, задающий параметры виртуальной среды для обучения агентов в задачах RL.

## **ВВЕДЕНИЕ**

В последние десятилетия наблюдается стремительное развитие индустрии цифровых развлечений, особенно в сфере видеоигр. Постоянно возрастающий интерес к играм сопровождается увеличением спроса на реализм и вовлеченность. Одним из наиболее эффективных способов достижения этих целей является создание игровых миров, которые функционируют, развиваются и реагируют на действия игрока. Этот аспект реализма может быть существенно усилен посредством внедрения искусственного интеллекта (ИИ) и методов машинного обучения.

Широкое распространение и доступность технологий искусственного интеллекта и машинного обучения в различных сферах человеческой деятельности стали явными факторами, привлекающими внимание как крупных корпораций, так и небольшие компании. В сфере создания видеоигр потенциал этих технологий пока не полностью реализован, но их применение обещает революционизировать игровой опыт.

В России и за рубежом было проведено значительное количество исследований, посвященных применению интеллектуальных алгоритмов для симуляции поведения врагов в играх. Многие из этих исследований сосредоточены на алгоритмах, связанных с детерминированным поведением игровых агентов, и стали важными источниками для данной диссертационной работы. Значительный вклад в теоретическую основу исследования внесли ученые Джулиан Т., Бенедикт М. Р. и Кристофер Х.

Целью данной магистерской диссертации является повышение реализма поведения противника в играх и уменьшение затрат на разработку ИИ-агентов для новых игр.

Объектом исследования является возможность применения нейронных сетей и методов машинного обучения при разработке игровых продуктов, а предметом – разработка программного обеспечения для симуляции поведения противников в видео играх.

Достижение цели магистерской диссертации включает в себя решение следующих задач:

- исследование и сравнительный анализ различных моделей нейронных сетей,
- исследование и сравнительный анализ инструментов создания компьютерных игр,
- создание тестовой среды для обучения игровых агентов,
- разработка программного обеспечения для загрузки и сохранения данных, необходимых для обученных моделей,
- создание среды для проверки эффективности обучения.

В качестве теоретической и методологической базы исследования выступают основные принципы математической статистики, линейной алгебры, методов оптимизации, а также исследования российских и зарубежных ученых в области машинного обучения, нейронных сетей и анализа данных.

# **1 Анализ предметной области**

## **1.1 Актуальность**

По данным World Economic Forum [1] индустрия игр в 2017 году оценивалась в 120 млн долларов. В 2024 году она достигла 278 млн. долларов. Эксперты прогнозируют рост этого рынка до 321 млн долларов к 2026 году. Рост связывают с распространением мобильных устройств и персональных компьютеров. Увеличение спроса на игровые продукты открывает новые возможности для создания уникальных проектов.

Многие игры созданы на основе противостояния компьютера игроку. Стандартный подход к созданию противников для игроков требует четкого описания всех возможных сценариев, что является сложным процессом. Ко всему прочему, это ограничивает вариативность поведения игрока в виртуальном мире. Применение машинного обучения и глубокого обучения предоставляет возможности для создания более адаптивных, вариативных и реалистичных ИИ-агентов, с которыми игроку было бы интересно взаимодействовать и противостоять.

Игры стремятся создать мир, в котором противники обладают собственной субъектностью и мотивацией, что подчеркивает необходимость разработки комплексных подходов к моделированию ИИ-агентов, включающих не только тактику или стратегию боя, но и социальные, эмоциональные и сюжетные аспекты.

Большинство игр концентрируются на применении ИИ для увеличения масштабов мира. Такой подход наполняет игру разнообразным бесконечным контентом. В качестве примеров можно привести такие проекты, как No Man's Sky и Starfield. Представленные проекты фокусируются на создании необычных локаций для игрока, причем каждая из них генерируется уникально и неповторимо. Разработчики заявляют о возможности создания еще более разнообразных миров [2, 3].

Стоит отметить, что эти проекты практически не используют ИИ для управления противниками или другими не игровыми персонажами, поэтому цепочки заданий часто повторяются.

Крупные производители графических ускорителей, такие как NVIDIA, активно инвестируют значительные ресурсы в разработку инфраструктуры для обучения ИИ-агентов. Одной из революционных технологий, представленных компанией, являются автономные игровые персонажи (Autonomous Game Characters, AGC). Данная технология позволяет моделировать и планировать поведение, приближенное к человеческому. Крупные игровые проекты, включая PUBG: BATTLEGROUNDS и inZOI, уже интегрировали эту технологию в свои игры [4].

Применение ИИ, в частности нейронных сетей, для моделирования поведения персонажей позволит совершить качественный скачок в создании реалистичных виртуальных миров.

## **1.2 Обзор методов создания искусственного интеллекта в играх**

Для того, чтобы понять целесообразность разработки системы, основанной на машинном обучении, требуется рассмотреть все самые используемые подходы к проектированию ИИ. К распространенным методам сознания игрового ИИ относятся следующие шаблоны проектирования:

- использования математической абстракции конечного автомата,
- использование дерева поведения,
- прямое кодирование ИИ,
- GOAP (Goal-Oriented Action Planning, англ. Целенаправленное планирование действий),
- машинное обучение и нейронные сети.

Каждый из указанных выше подходов имеет широкое распространение и свои достоинства и недостатки, для полноты анализа требуется рассмотреть каждый из методов проектирования в отдельности [3].

### 1.3 Прямое кодирование ИИ

Написание игрового ИИ без использования шаблонов представляет собой подход, при котором разработчики создают алгоритмы искусственного интеллекта с нуля. Применение этого подхода в играх имеет ряд преимуществ и недостатков. К достоинствам можно отнести:

- возможность точно настроить поведение ИИ под поставленную задачу,
- позволяет быстро реализовать поведение без значительных временных затрат,
- эффективен при разработке небольших проектов с достаточно малым числом взаимодействия пользователя с ИИ,
- избегание шаблонов может позволить лучше оптимизировать код для конкретной игры или среды разработки, учитывая ее уникальные характеристики и требования к производительности.

К недостаткам данного подхода можно отнести следующие характеристики:

- использование подхода увеличивает вероятность возникновения ошибок в сложных сценариях,
- в больших проектах данный подход усложняет отладку и тестирование всей игры,
- сложность документирования проекта,
- качество и эффективность кода сильно зависит от навыков человека реализующего ИИ,
- отсутствие стандартных шаблонов может привести к потере некоторых важных аспектов или созданию уязвимостей в поведении ИИ, что потенциально может повлиять на игровой баланс,
- поддержка и обновление кода является сложной задачей, особенно это заметно при необходимости добавления новых персонажей или адаптации к изменениям в игре,

- метод поощряет дублирование кода,
- разработанное решение сложно использовать повторно или интегрировать в другую игру.

Разработка искусственного интеллекта без использования шаблонов требует баланса между творчеством и практичностью разработчика. Данный подход применим для небольших игр с ограниченным числом взаимодействий с ИИ. Однако его сложно использовать при часто меняющихся требованиях к игре и к поведению противника [4].

#### **1.4 Использование шаблона конечного автомата**

Шаблон конечного автомата (Finite State Machine, FSM) [5] представляет собой распространенный подход к проектированию и реализации искусственного интеллекта в играх на основе математического аппарата графов. Внедрение FSM в разработку ИИ позволяет легко моделировать различные состояния и переходы между ними, создавая гибкую систему управления поведением противников и не игровых персонажей. Основные преимущества этого шаблона:

- FSM позволяет разделить поведение персонажа на отдельные состояния, что делает процесс проектирования и кодирования более простым и структурированным,
- структура конечного автомата интуитивно понятна, что упрощает как сам процесс разработки, так и отладку кода, это позволяет разработчикам легко отслеживать состояния и переходы, облегчая диагностику проблем,
- конечные автоматы обеспечивают легкость внесения изменений в поведение персонажей, а добавление новых состояний или модификация существующих может быть осуществлено без серьезных изменений в других частях кода,
- FSM хорошо подходят для моделирования сценариев средней сложности, где персонаж может находиться в различных состояниях в ответ на внешние воздействия,

- каждое состояние и соответствующие ему переходы выражены явно, что упрощает восприятие кода другими разработчиками,
- большое количество готовых решений под разные платформы и среды разработки,
- подход знаком многим разработчикам, что дает возможность поддерживать и вносить изменения разным людям в команде,
- FSM могут быть эффективными с точки зрения использования ресурсов, особенно в сравнении с более сложными методами моделирования поведения, такими как машинное обучение.

Несмотря на множество преимуществ, использование шаблона конечного автомата в разработке искусственного интеллекта для игр также сопряжено с определенными ограничениями:

- конечный автомат может оказаться недостаточным для моделирования крайне сложных и разнообразных сценариев поведения,
- в некоторых случаях количество состояний и переходов может неконтролируемо расти, что приводит к увеличению сложности и уменьшению читаемости кода,
- конечные автоматы являются памятью без состояния, что может быть недостаточным для определения и запоминания долгосрочных или сложных взаимосвязей в поведении противника или группы ИИ-агентов,
- возможность возникновения петель в состояниях, что может привести к зависанию как персонажа, так и всей игры,
- сложность реализации взаимодействия между разными агентами,
- FSM не всегда являются лучшим выбором для моделирования параллельных или одновременно выполняющихся задач, также обработка нескольких состояний одновременно может потребовать дополнительных усилий,

- при создании сложных состояний и переходов может потребоваться значительное количество кода, что, в свою очередь, может увеличивать вероятность ошибок и усложняет поддержку кода,

В целом, использование шаблона конечного автомата в создании искусственного интеллекта для игр представляет собой мощный инструмент, обеспечивающий гибкость и эффективность в реализации различных типов поведения противников и персонажей. Подход эффективно использовать в играх средней сложности и с малым количеством возможных состояний.

### **1.5 Дерево поведения**

Как и для любого инструмента для проектирования и разработке игрового ИИ, шаблон дерева поведения имеет свои достоинства и недостатки [6].

К достоинствам метода можно отнести следующие пункты:

- деревья поведения позволяют создавать модульные структуры для управления поведением персонажей, каждая ветвь дерева может представлять собой отдельное поведение, что упрощает добавление и изменение функциональности,
- деревья поведения легко визуализировать, что облегчает понимание и отладку системы управления, при этом разработчики могут четко видеть, какие решения принимает искусственный интеллект в различных ситуациях,
- добавление новых поведений к персонажу часто ограничивается созданием новой ветви в дереве, что существенно упрощает процесс расширения функциональности ИИ,
- большое количество готовых решений для разнообразных сред разработки,
- адаптивность к сложным сценариям – деревья поведения предоставляют удобный способ моделирования сложных сценариев, включая ситуации с несколькими вариантами выбора,

- возможность добавления визуального программирования для создания игрового ИИ,
- состояния и целые ветви решающего дерева можно переносить и смешивать, что дает дополнительную вариативность в поведении,
- параллельные задачи могут быть эффективно моделированы в деревьях поведения, что делает их удобными для сценариев, где персонаж должен выполнять несколько задач одновременно.

К минусам можно отнести следующие характеристики:

- в сложных деревьях поведения может возникнуть трудности в их проектировании и понимании, требуется внимательное планирование и строгое техническое задания, описывающего все взаимодействия,
- разветвленные и глубокие решающие деревья могут потребовать значительных вычислительных ресурсов, что может отрицательно сказаться на производительности игры, особенно на устройствах с ограниченными ресурсами,
- в некоторых случаях структура дерева может привести к неоднозначности в принятии решений, особенно если не проводится тщательное тестирование различных сценариев,
- сложность добавления взаимодействия между агентами,
- добавление нового поведения требует создания ветки поведения, что может привести к запутанности в больших изменяющихся проектах.

Очевидно, что многие достоинства и недостатки данного метода аналогичны шаблону конечного автомата, поскольку в основе обоих подходов к проектированию ИИ лежит единый математический аппарат теории графов. Следует также отметить, что дерево поведения обладает более строгой структурой по сравнению с традиционным конечным автоматом. Визуальные различия между этими двумя подходами представлены на рисунках 1 и 2.

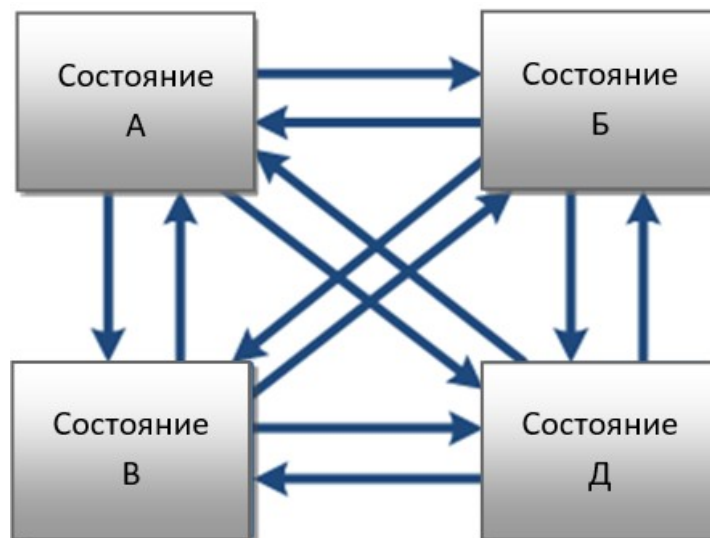


Рисунок 1 – Конечный автомат с 4 состояниями

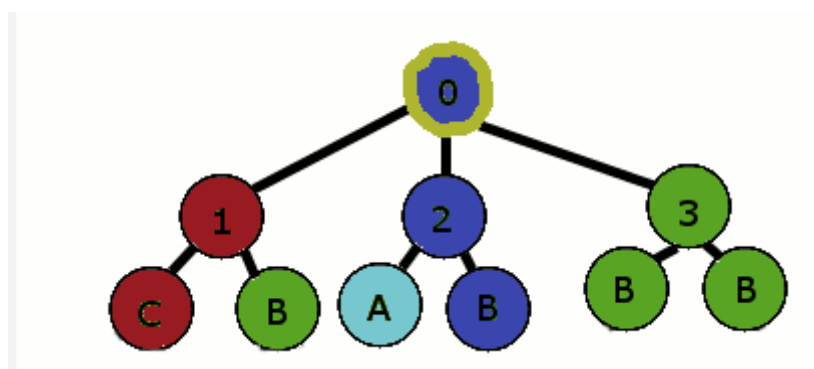


Рисунок 2 – Дерево поведения.

Главное отличие – это то, что в модели конечного автомата, все состояния связаны друг с другом, что делает программу сильно связанной.

В дереве поведения процесс принятия решения ИИ более структурирован. На рисунке красным цветом показаны узлы, которые не смогли выполняться из-за предусловий, зеленым отмечены узлы готовые к исполнению, голубым – выполненные, а синем выполнявшиеся.

## 1.6 GOAP подход

GOAP – подход к созданию игрового ИИ, использующий систему планирования действий для достижения целей. Метод оценивает доступность действий и возможный их результат. Данный метод часто применяется в высоко бюджетных игровых проектах, где требуется реалистичность.

Рассмотрим основные сильные и слабые стороны этого метода проектирования [7].

Рассмотрим преимущества этого метода разработки ИИ:

- метод направлен на достижение цели и позволяет ИИ самостоятельно находить оптимальные пути решения задачи,
- агенты выбирают действия динамически исходя из текущей конъюнктуры игрового мира и внутренних состояний, что позволяет агенту выглядеть более естественно и реалистично,
- действия в этом подходе легко переиспользовать и модифицировать, поскольку они являются атомарными.

Рассмотрим самые значительны узкие места данного метода:

- для использования этого метода от разработчика требуется большой опыт в области создания ИИ и умение применять именно этот подход,
- сложность отладки причинно-следственных связей, при принятии решений,
- подходит только для сложных проектов, т.к реализация занимает много времени,
- не эффективен при решении простых целей и задач.

GOAP фокусирующийся на достижении целей через генерацию и выполнение планов действий, что позволяет методу достигать комплексных целей. При этом использование этого подхода для реализации простых действий неэффективно.

### **1.7 Нейронные сети и машинное обучение**

Методы машинного обучения и нейронных сетей можно поделить на несколько категорий. Каждую категорию можно так или иначе использовать для проектирования игрового ИИ. К таким методам отнести следующие:

- обучение с учителем,
- обучение с подкреплением,

- генетические алгоритмы,
- глубокие нейронные сети,
- рекуррентные нейронные сети.

Для наглядности рассмотрим каждый из методов и область их применения в рамках искусственного интеллекта для игр.

Обучение с учителем можно использовать для анализа данных об игровом процессе. Метод обучения может быть эффективно использован для игр жанра Computer Role-Playing Game (RPG, англ. Компьютерная ролевая игра), где требуется планировать действия агента на несколько ходов вперед при этом отталкиваясь от истории поведения оппонента.

Метод обучения с подкреплением целесообразно применять в случаях, когда для обучения ИИ противника недостаточно данных или они отсутствуют. Система вознаграждений и штрафов позволяет корректировать поведение противника, и изменять его в требуемом направлении. Этот подход эффективен для игр с возможностью многократного прохождения одних и тех же уровней, что позволяет как игроку, так и ИИ адаптировать свое поведение в зависимости от каждого прохождения [8].

Генетические алгоритмы могут быть использованы для плавного повышения сложности игры. В данном случае оптимальные стратегии передаются между поколениями противников, что обеспечивает их постепенное усложнение. Следует отметить, что этот метод наиболее эффективен для моделирования поведения простых противников.

Глубокие нейронные сети целесообразно применять в сложных системах, моделирующих обширные сети взаимосвязей между персонажами или сложные экономические системы. Это наиболее сложный в реализации вариант игрового ИИ, поскольку внедрение таких нейронных сетей может привести к нарушению игрового баланса и непредсказуемому поведению противников. Данный метод позволяет обучать группу противников как

единую взаимодействующую систему, что обеспечивает реализацию сложных координированных стратегий.

Рекуррентные нейронные сети эффективны для моделирования последовательностей действий противников во временном контексте. Это особенно полезно в играх, где противники должны адаптироваться к действиям игрока и изменениям игровой среды.

На основании проведенного анализа можно выделить ключевые достоинства и недостатки рассматриваемого метода проектирования. Можно выделить несколько преимуществ :

- обученные с использованием нейронных сетей противники способны адаптироваться к действиям игроков и изменениям игровой среды, что повышает их интеллектуальный уровень и сложность поведения,
- нейронные сети позволяют создавать более сложные стратегии противодействия, а также освобождает разработчиков от необходимости предусматривать все возможные варианты поведения игрока,
- нейронные сети позволяют моделировать поведение противников, приближенное к реальному, что особенно важно для большинства компьютерных игр,
- модели могут совершенствоваться на основе накопленного опыта, что позволяет оптимизировать игровые стратегии и экономический баланс .

К основным недостаткам данного вида проектирования ИИ можно отнести:

- сложность реализации,
- непредсказуемость поведения,
- проблемы балансировки.

Сложность реализации – создание сложных моделей представляет значительные трудности, особенно для небольших команд разработчиков или проектов с ограниченными сроками. Дефицит специалистов, обладающих

опытом как в разработке игр, так и в работе с нейронными сетями, существенно затрудняет процесс разработки.

В большинстве игровых сценариев настраивается сложная математическая модель, где неопределенность и непредсказуемость нейронных сетей могут быть нежелательными. Это может привести к ограничению игровых возможностей или выходу за рамки этических норм [9].

Проблемы балансировки – традиционно баланс настраивается на основе человеческого игрового опыта. Для его динамической настройки требуется сложная система ограничений, которые часто плохо интерпретируются или имеют сильную зависимость от специфики проекта.

Применение нейронных сетей для моделирования поведения противников в играх требует тщательного баланса между качеством игрового процесса и адаптивностью стратегий. Данные методы продолжают развиваться, и разработчики постоянно исследуют новые подходы к совершенствованию игр с использованием искусственного интеллекта.

Следует отметить, что как игровая индустрия, так и технологии нейронных сетей развиваются стремительными темпами, а потенциал применения нейронных сетей в компьютерных играх еще далеко не исчерпан.

## 2 Выбор средств разработки

Для реализации проекта важно выбрать язык программирования, а также среду создания игр, которые наилучшим образом соответствует задачам. Рассмотрим несколько возможных вариантов интегрированных средств создания игр.

### 2.1 Unreal Engine и C++

Unreal Engine (UE) – крупное средство для создания игр и интерактивных приложений. Рассмотрим плюсы и минусы этого средства.

К ключевым достоинствам UE можно отнести следующие инструменты:

- встроенные инструменты для создания реалистической графики в играх,
- большое количество инструментов и ресурсов для создания игр,
- поддержка мультиплатформенности, дает возможность запускать проект на разных архитектурах ЭВМ,
- бесплатная лицензия до заработка проекта до 1 млн. долларов,
- есть интеграция с PyTorch через API.

К недостаткам UE можно отнести следующие особенности:

- высокий порог входа для работы с инструментами UE,
- размер проекта и оптимизация, для создания оптимизированных локаций и игр, требуется большое количество времени,
- инструменты интеграции с PyTorch и библиотеками для машинного обучения находятся на этапе создания и тестирования.

В UE основным языком программирования является C++ 20, что увеличивает порог входа для новых разработчиков, а также затрудняет создание проектов. Среда имеет проблемы с совместимостью некоторых стандартных библиотек языка. Также UE не поддерживает контейнеры, такие как `std::map`, `std::vector` и другие. Некоторые синтаксические элементы,

например, лямба-выражения, шаблоны, рефлексия не работают в контексте Unreal Reflection System [10], что является большим недостатком.

## 2.2 Unity и C#

Unity – программное обеспечение для создания интерактивных приложений и игр. Оно приобрело большую популярность в сфере разработке игр из-за своей доступности и дружелюбности для начинающих. Рассмотрим несколько ключевых достоинств Unity:

- простой дружелюбный интерфейс,
- большое количество обучающих материалов,
- среда позволяет создавать приложения под все виды популярных устройств,
- большое сообщество разработчиков,
- интеграция с современными IDE, такими как Rider, Visual Studio,
- поддержка API для интеграции TensorFlow, PyTorch и других библиотек,
- низкие системные требования позволяют запускать среду почти на всех устройствах,
- благодаря своей легкости Unity позволяет создавать готовые решения за короткий срок.

К минусам можно отнести следующие особенности:

- бесплатная версия показывает начальный экран с водяным знаком Unity при запуске игры, что может быть недостатком для профессиональных разработчиков,
- профессиональная версия Unity стоит дороже аналогов, что может быть проблемой для небольших команд.
- ограничения для реализации фото реалистичной графики.

C# – это объектно-ориентированный язык с простым синтаксисом, что делает его лёгким для изучения и использования. C# поддерживает асинхронность с помощью `async/await`. Все стандартные коллекции доступны

в Unity, а также широкие возможности для программиста дают инструменты такие как: лямбда-выражения, шаблоны, LINQ, большое количество совместимых библиотек. Среды разработки C# предоставляют мощные средства отладки, что ускоряет процесс разработки [11].

## 2.3 Godot Engine

Godot Engine (GE) – среда разработки игр с открытым исходным кодом, активно развивающиеся последние несколько лет.

Можно выделить следующие достоинства GE:

- полностью бесплатная среда разработки, распространяемая по лицензии MIT,
- кроссплатформенность – GE поддерживает создание игр для следующих платформ Windows, MacOS , Android, IOS, HTML5,
- простота использования широких функций среды разработки,
- производительность – благодаря модульной структуре любого проекта, приложение можно просто оптимизировать,
- поддержка нескольких языков программирования, позволяет просто перейти на новую среду.

Минусы Godot:

- слабо развито 3D и визуальное составляющее проектов в этой среде,
- относительно новая экосистема, многие типичные проблемы еще не исправлены,
- нет инструментов для VR/AR,
- нет достаточно функций для созданий сетевых игры,
- нет поддержки API для взаимодействия с Python-библиотеками машинного обучения.

Godot поддерживает GDScript C#, C++, VisualScript и Python, но не в каждом языке реализованы весь функционал и можно осуществить доступ ко

всем библиотекам. Самый многофункциональными и с поддержкой наибольшего числа API являются Python и GDScript [12].

## 2.4 Критериальная оценка сред разработки игровых приложений

Для проведения критериальной оценки ПО, следует выделить основные метрики оценки и веса за каждый критерий. Критерий оценки и максимальный возможный балл за критерий представлен в таблице 1.

Таблица 1 – Баллы по критериям

| Критерий                                              | Минимальная оценка | Максимальная оценка |
|-------------------------------------------------------|--------------------|---------------------|
| Лицензионное соглашение и тарифы на ПО                | 0                  | 0.5                 |
| Удобство API взаимодействия с библиотеками МО         | 0                  | 1.5                 |
| Простота разработки                                   | 0                  | 1                   |
| Качество документации                                 | 0                  | 1                   |
| Скорость разработки проектов                          | 0                  | 1                   |
| Используемый язык программирования в среде разработки | 0                  | 1                   |

Для определения наиболее подходящей среды разработки в таблице 2 сопоставлены веса и среды разработки.

Таблица 2 – Оценка сред разработки по выбранным критериям

| Критерий                                              | Unity | Unreal Engine | Godot Engine |
|-------------------------------------------------------|-------|---------------|--------------|
| Лицензионное соглашение и тарифы на ПО                | 0.1   | 0.4           | 0.5          |
| Удобство API взаимодействия с библиотеками МО         | 1.3   | 0.7           | 0.5          |
| Простота разработки                                   | 1     | 0.65          | 1            |
| Качество документации                                 | 0.8   | 0.7           | 0.3          |
| Скорость разработки проектов                          | 0.7   | 0.2           | 0.5          |
| Используемый язык программирования в среде разработки | 0.9   | 0.3           | 0.8          |

Для определения, наиболее подходящего средства разработки игр, все баллы за выбранные метрики были просуммированы. Результаты проведения критериального анализа представлены в таблице 3.

Таблица 3 – Средний балл по выбранным критериям

| Средство разработки | Суммарный балл |
|---------------------|----------------|
| Unity               | 4.8            |
| Unreal Engine       | 2.95           |
| Godot Engine        | 3.6            |

В ходе анализа было выявлено, что наиболее функциональной средой разработки игр по выбранным критериям является Unity. Критерии были выбраны таким образом, чтобы можно было выявить среду, которая позволяет достичь высокой скорости разработки и возможности интеграции нейронных сетей в игру.

## 2.5 Выбор интегрированной среды разработки

После выбора среды разработки игры, требуется выбрать интегрированную среду разработки для написания кода (integrated development environment, IDE). Unity поддерживает несколько IDE для написания и отладки кода, например: Microsoft VisualStudio, Rider, MonoDevelop.

Microsoft VisualStudio – популярная среда разработки от Microsoft, предлагающие мощные инструменты для разработки в Unity [13].

Достоинства IDE:

- поддержка Unity и отладки в процессе выполнения программы,
- автодополнение и продвинутая система подсказок,
- большое количество плагинов и расширений,
- обширная документация и поддержка сообщества.

Недостатки IDE:

- для начинающих интерфейс программы – избыточен и перегружен,

- большая ресурсоёмкость.

JetBrains Rider – IDE от компании JetBrains предназначенное для профессиональных разработчиков [14].

Главные достоинства:

- производительность – быстрее чем у Visual Studio, особенно в крупных и масштабных проектах,
- интеграция с Unity и поддержка отладки,
- умные подсказки и автодополнения,
- кроссплатформенность.

К основным недостаткам можно отнести:

- платная версия,
- сложность для начинающих.

MonoDeveloper – IDE которая ранее была частью Unity, сейчас теряет популярность [15].

Достоинства IDE:

- производительность – IDE легковесная и может запускаться на любом ЭВМ,
- хорошая интеграция с Unity.

Недостатки:

- поддержка только от сообщества,
- нет мощных инструментов разработки,
- теряет популярность у сообщества.

Для реализации проекта была выбрана Microsoft VisualStudio как наиболее оптимальная IDE. Ключевыми характеристиками при выборе этой IDE, стали ее доступность и высокий уровень автоматизации и интегрированности в среду Unity.

### **3 Постановка задачи на разработку**

#### **3.1 Анализ существующих решений**

Для анализа актуальности и экономической эффективности разработки ИИ, моделирующего поведение противников, целесообразно обратиться к данным официального портала государственных закупок. Данный ресурс был выбран по следующим причинам:

- как источник информации о конкурентных решениях на рынке,
- сайт государственных закупок позволяет проанализировать основные параметры аналогичных решений и их спецификации,
- позволяет выявить тенденции и перспективные направления в которые государство готово инвестировать средства,
- как потенциальный канал реализации продукта через систему государственного заказа.

Моделирование сложного поведения представляет собой распространенную задачу в сфере прогнозирования и управления государственными ресурсами. Анализ подобных заказов позволяет оценить степень востребованности нейросетевых технологий при решении данного класса задач.

В качестве примера рассмотрим конкретный объект закупки (рисунок 3): «Прототип цифровой платформы моделирования транспортного поведения с апробацией на расширенной структуре исходных данных для решения задач прогнозирования транспортного спроса на макроуровне» [16]. Данный проект демонстрирует практическое применение технологий моделирования сложного поведения в государственном секторе.

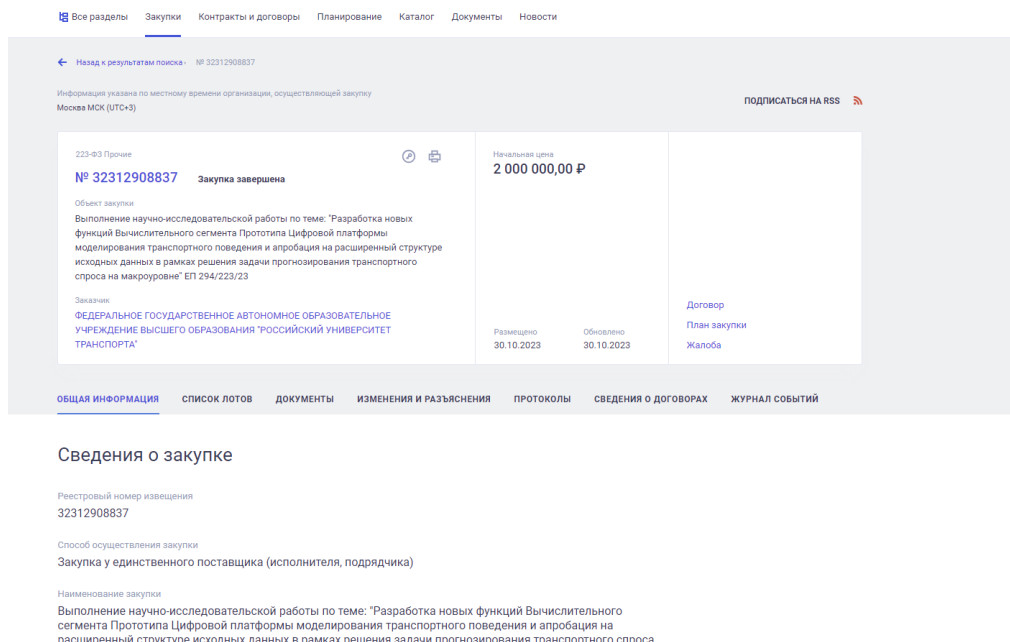


Рисунок 3 – Сведения об объекте закупки

Рассмотрим основные функции системы. К ним можно отнести:

- визуализация распределения автомобильных потоков для разных классов пользователей,
- формирование матрицы обобщённых затрат между пунктами транспортной сети,
- реализация многослойного картографического интерфейса,
- расчёт суммарных объёмов отправления и прибытия по районам.

Для реализации этих функциональных особенностей требуется провести следующие исследования :

- обобщение модели для множества классов пользователей,
- адаптация алгоритмов поиска равновесия для моделей Бекмана и Стабильной динамики,
- интеграция вычислительного сегмента в цифровую платформу,
- тестирование взаимодействия с пользовательским сегментом,
- валидация на расширенном наборе исходных данных.

Указанная система должно уметь обрабатывать следующие параметры:

- географические координаты узлов сети,

- весовые коэффициенты рёбер,
- разрешённые направления движения,
- количество полос движения по направлениям,
- пропускная способность полос,
- ограничения для категорий ТС,
- наличие специализированных полос,
- ограничения скорости,
- организация движения на перекрёстках.

В средстве должны быть реализована возможность модификации следующих входных данных:

- указание направлений для дуг связи,
- списки доступных транспортных мод,
- индивидуальные матрицы затрат,
- учёт габаритно-динамических коэффициентов.

ПО должно в полной мере реализовывать требования к обмену данными:

- использовать протокол grps,
- передавать входных данные,
- передавать результаты отчетов.

Из технического задания можно выделить основные цели разработки ПО:

- прогнозирование пассажиропотоков,
- расчёт транспортных показателей,
- оценка экономической эффективности,
- анализ экологических последствий.

Анализ объекта закупки свидетельствует о том, что разрабатываемое решение может быть применено не только для моделирования поведения в играх, но и для моделирования поведения людей в сфере государственного управления.

В качестве второго решения, следует рассмотреть программное обеспечение позволяющие реализовать поведение противников в играх.

Одним из решений на рынке, позволяющему реализовывать нейронные сети в проектах является ANN Perceptron [17].

Программное средство позволяет обучать нейронные сети в Unity. Плагин позволяет создать перестроен с заданным количеством нейронов, а также просматривать веса нейронной сети.

Плагин интегрирован с unity и позволяет обучать нейронную сеть на центральном процессоре (ЦПУ). Графический интерфейс расширения позволяет редактировать нейронные связи.

К основным преимуществам этого программного средства можно отнести:

- высокая степень интеграция с Unity,
- визуализация архитектуры нейронной сети,
- возможность ручного изменения весов,
- поддержка обучения агентов.

К недостаткам можно отнести:

- поддерживаются только 3-слойные сети,
- невозможно увеличить число скрытых слоёв или нейронов,
- обучение только на ЦПУ, без поддержки графического процессора (ГПУ),
- нет распределённого обучения на нескольких машинах,
- сложность отладки из-за закрытости алгоритмов,
- отсутствие настройки гиперпараметров.

Данный плагин подходит для создания простых поведенческих моделей и быстрого обучения агентов, используя метод обратного распространения ошибки. Однако он серьёзно ограничен в гибкости: нельзя

модифицировать процесс обучения, изменить структуру сети или масштабировать вычисления. Это делает его неприменимым для сложных сценариев ИИ в современных играх.

В магазине Asset Store для Unity доступно гибкое и практичное расширение Simple Neural Network [18]. Система предоставляет возможность создания многослойных нейронных сетей и сохранения их коэффициентов в файл. В качестве сопровождающей документации и демонстрации работоспособности системы программное обеспечение включает проекты по симуляции поведения машин и распознаванию рукописных цифр.

Несомненным преимуществом проекта является возможность его применения не только для обучения ИИ-агентов, но и для решения более общих задач. Плагин отличается высокой степенью интеграции с Unity, хорошо проработанной документацией и визуальным отображением нейронных сетей. Для обучения система использует алгоритм обратного распространения ошибок.

К основным преимуществам данной системы можно отнести:

- высокую степень интеграции с Unity,
- визуальное отображение обученных сетей,
- сохранение параметров обучения,
- широкий спектр задач, которые может решать система.

В то же время система имеет ряд недостатков:

- обучение нейронных сетей осуществляется только ЦПУ,
- обучение возможно только в одной среде,
- невозможность распараллеливания вычислений,
- сложность отладки и модификации,
- доступность только одного алгоритма обучения,
- трудность в обучение сложных нейронных сетей,
- сложность интеграции в проекты,

- отсутствие возможности изменения гиперпараметров.

Данная система позволяет применять нейронные сети для обучения ИИ-агентов с ограниченным набором доступных действий. Однако производительность системы ограничена, поскольку реализация нейронных сетей выполнена на языке C#, что затрудняет распараллеливание кода.

Проанализировав доступные программные решения, были выявлены следующие общие недостатки в каждой системе:

- отсутствие возможности обучения нейронных сетей на графическом процессоре (ГПУ),
- сложность отладки и обучения,
- сложность интеграции в существующие проекты,
- ограниченный набор алгоритмов обучения,
- невозможность изменения гиперпараметров.

В связи с этим было принято решение разработать систему обучения ИИ-агентов для Unity.

### **3.2 Цель и назначение системы**

При разработке крупных игровых проектов традиционные подходы к созданию искусственного интеллекта, такие как деревья поведения или планирование на основе целей, сталкиваются с рядом существенных ограничений. Основная проблема заключается в стремительном усложнении кодовой базы при реализации сложных поведенческих моделей, что негативно сказывается на сопровождаемости системы. По мере расширения функциональности существенно возрастают временные затраты на добавление новых поведенческих шаблонов, при этом процесс полнофункционального тестирования приобретает чрезмерную трудоемкость.

Современные системы машинного обучения, применяемые для разработки игрового ИИ, демонстрируют недостаточную гибкость и не

предоставляют удобных инструментов для быстрой интеграции. Это создает значительные проблемы для эффективного внедрения алгоритмов в игры.

Разрабатываемое программное решение нацелено на оптимизацию ключевых аспектов создания игрового ИИ, включая процессы обучения, интеграции и тестирования. Внедрение ПО позволит достичь сразу двух целей: повышения экономической эффективности разработки за счет сокращения временных затрат и одновременного улучшения качества моделирования поведения не игровых персонажей.

Главной задачей проекта является создание высокоэффективного инструментария для обучения игрового ИИ, сопровождаемого демонстрационным проектом. Прототип послужит валидации работоспособности предложенного решения, так и для наглядного пособия его возможностей потенциальным пользователям. Особое внимание уделяется обеспечению простоты интеграции и гибкости настройки, что должно сделать проект привлекательным для разработчиков разного уровня подготовки.

### **3.3 Требования к программе**

Разрабатываемое программное обеспечение представляет собой библиотеку в виде DLL-модулей, предназначенных для интеграции с Unity Editor и Unity Runtime. Основная задача системы заключается в обеспечении процесса обучения ИИ-агентов в специально созданных тестовых средах с возможностью последующего использования обученных моделей в игровых проектах.

Архитектурно решение будет построено на принципах модульности и гибкости, позволяя работать как в среде разработки Unity Editor так и в конечном продукте. Для хранения результатов обучения используется YAML-формат данных, обеспечивающий совместимость и переносимость моделей между различными проектами.

Функциональные возможности включают создание специализированных тестовых уровней, где ИИ-агенты могут обучаться через систему вознаграждений. Агенты поддерживают заданный набор действий, которые могут быть адаптированы под конкретные игровые сценарии. Важной особенностью является предоставление инструментов для тонкой настройки гиперпараметров нейронной сети и отслеживания истории обучения.

Техническая реализация подчеркивает важность производительности и кросс-платформенной совместимости. Особое внимание уделено простоте интеграции в существующие проекты. Решение предлагает разработчикам эффективный инструмент для создания адаптивных ИИ-систем, значительно сокращающий временные затраты по сравнению с традиционными подходами к разработке игрового ИИ.

### **3.4 Функциональные требования**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- обучение ИИ-агентов в Unity Editor,
- сохранение результатов обучения модели в YAML-файлы,
- загрузка результатов обучения из YAML-файлов,
- использование обученных моделей в конечных программных средствах,
- обеспечивать программный интерфейс для взаимодействия с Tensor Board,
- сохранять история обучения нейронной сети,
- позволять редактировать гиперпараметры.

Указанные функции позволят обучать и вводить в эксплуатацию нейронные сети в проект любой сложности.

## 4 Разработка системы

### 4.1 Разработка расширения для Unity

В Unity есть возможность обучать нейронные сети и применять результат их обучения в играх. Взаимодействие Unity и TensorFlow [19] для обучения противников представлено на диаграмме компонентов на рисунке 4.

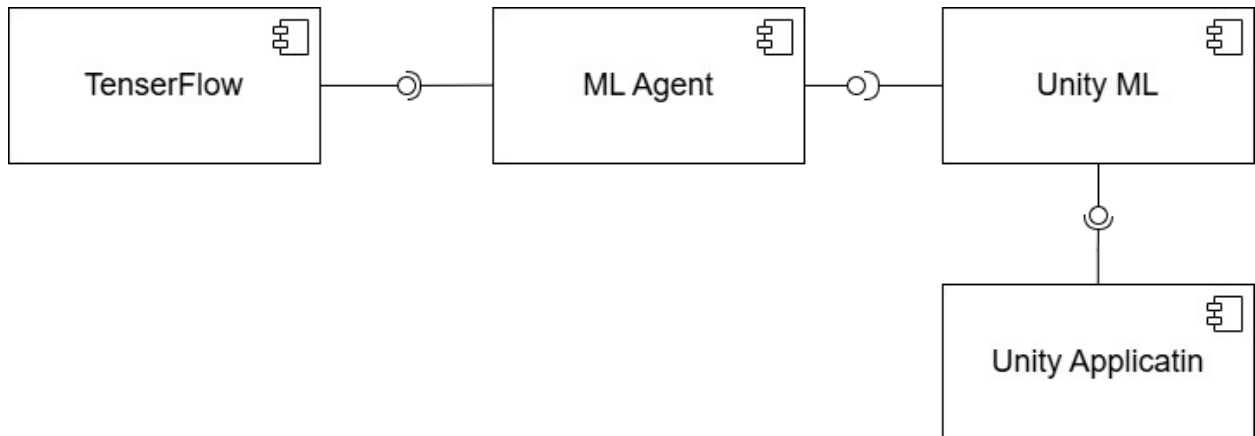


Рисунок 4 – Диаграмма компонентов

Для обучения нейронной сети необходимо наличие интегрированного в проект API на C#, обеспечивающего взаимодействие с пользовательским кодом. На диаграмме этот элемент обозначен как компонент «Unity ML». Пользовательский код взаимодействует с консольной утилитой, представленной компонентом «ML Agent», который, в свою очередь, обеспечивает интеграцию с библиотекой TensorFlow на Python. После обработки данных ML Agent получает результаты корректировки весов модели, сохраняемые в файл формата ONNX. Схема жизненного цикла обучения модели отображена на диаграмме последовательностей (рисунок 5).

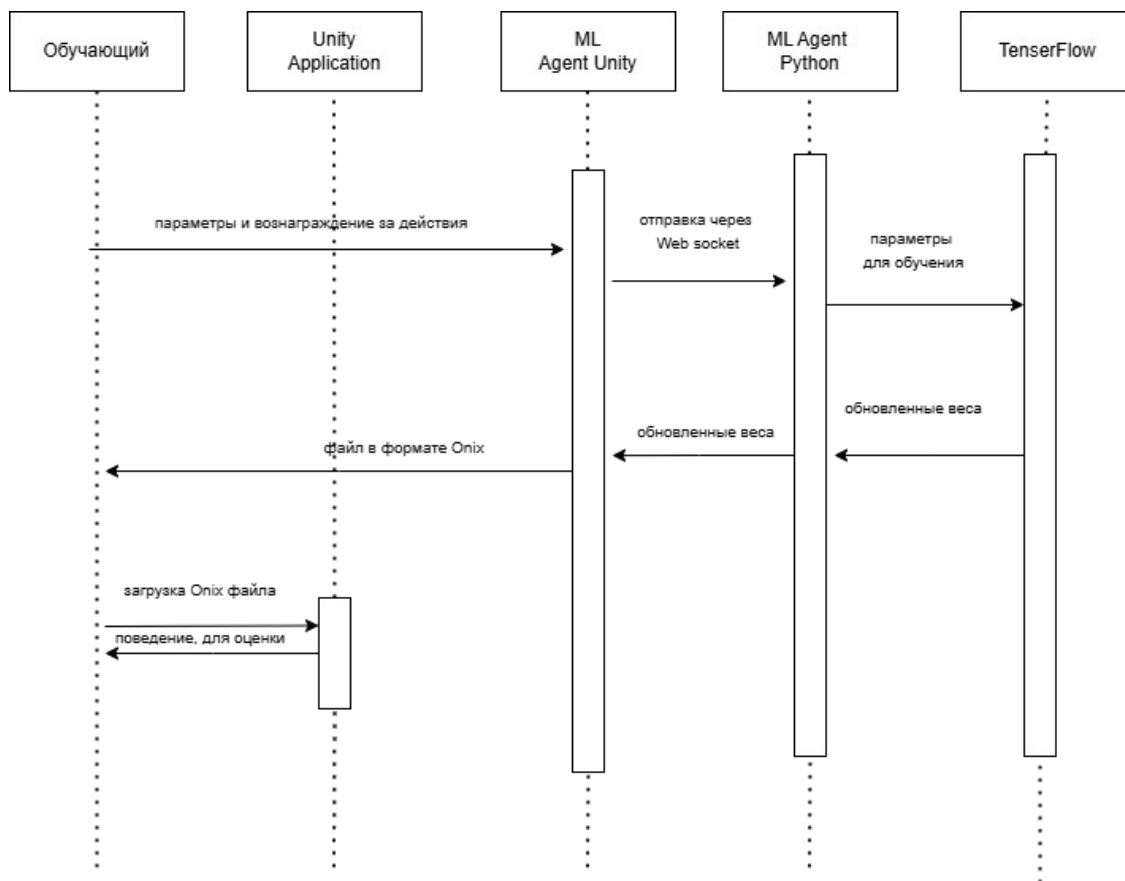


Рисунок 5 – Диаграмма последовательности.

В текущей реализации отсутствуют инструменты оперативной корректировки поведенческих параметров и системы вознаграждений. Существенной проблемой остается работа с ONNX-файлами, содержащими обученные веса нейронной сети – процедуры их экспорта и импорта требуют значительных временных затрат. Кроме того, текущая архитектура взаимодействия компонентов вынуждает выполнять многократное повторение полного цикла для тонкой настройки модели.

Ключевым ограничением интеграции TensorFlow с Unity выступает необходимость ручного управления процессами через командную строку (CMD). Например, для запуска обучения в шести параллельных средах с использованием предобученной модели T\_27\_04 и аппаратного ускорения CUDA требуется явное выполнение команды листинге ниже:

```
mlagents-learn config.yaml --run-id=T_27_04
--env="Build/MGA.exe" --num-envs=6 --no-graphics --torch-device
cuda
```

Дополнительную сложность создает требование ручной настройки окружения для каждой сборки для обучения нейронной сети. Для минимизации временных затрат реализована автоматическая генерация тестовых сцен системой.

Указанные проблемы решены за счет реорганизации схемы взаимодействия компонентов, устраняющей избыточные этапы и исправляющий основные недостатки API при обучении нейронных сетей (рисунок 6).

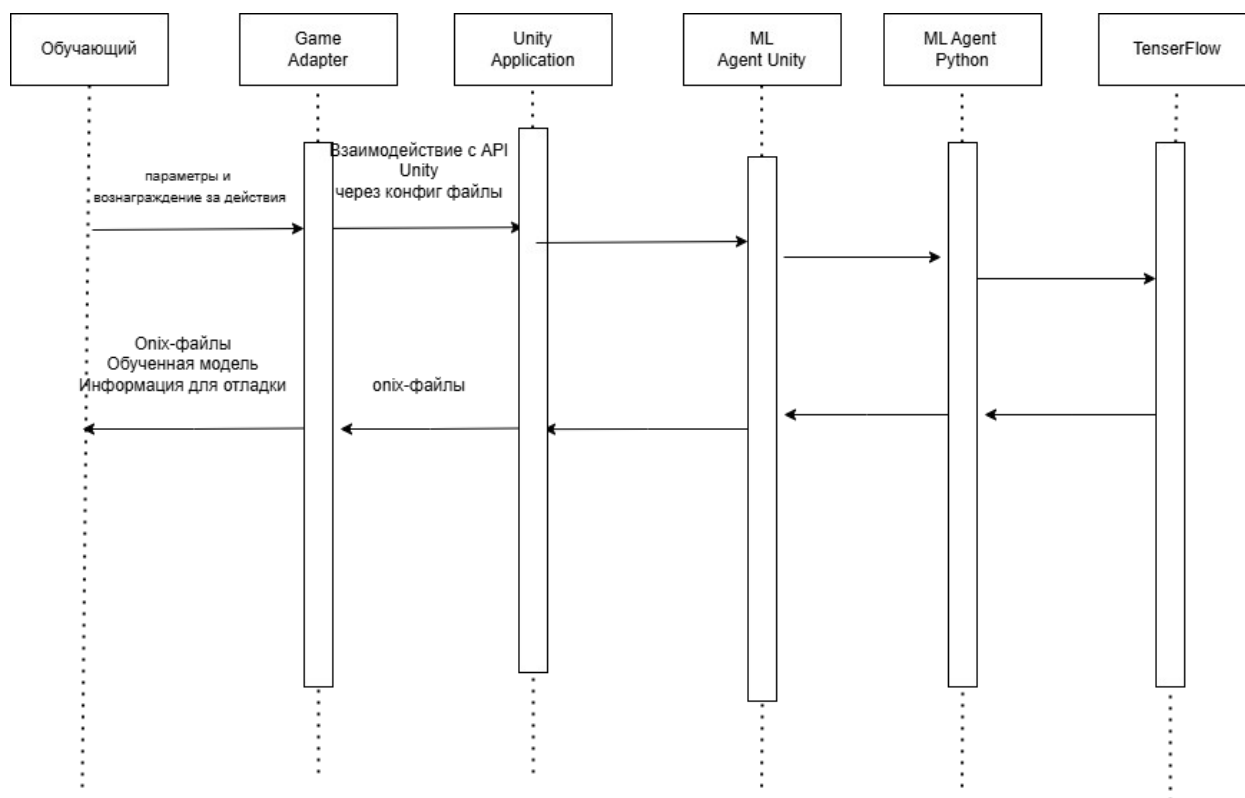


Рисунок 6 – Диаграмма последовательности.

В реализованной функциональности «GameAdapter» берет на себя функцию сохранения последовательностей обучения и параметров моделей, формирует информацию для отладки поведения модели, а также сразу же интегрирует модель в прикладной код. Интеграция этого адаптера позволит

ускорить процесс обучения модели и разработки конечного программного продукта.

Базовым компонентом Unity ML выступает класс Agent, реализующий методы обработки параметров, прогнозируемых нейронной сетью, а также способный генерировать вознаграждения за корректные действия агента. Для проектирования системы разработана диаграмма классов, отражающая ключевые структурные элементы программы (рисунок 7).

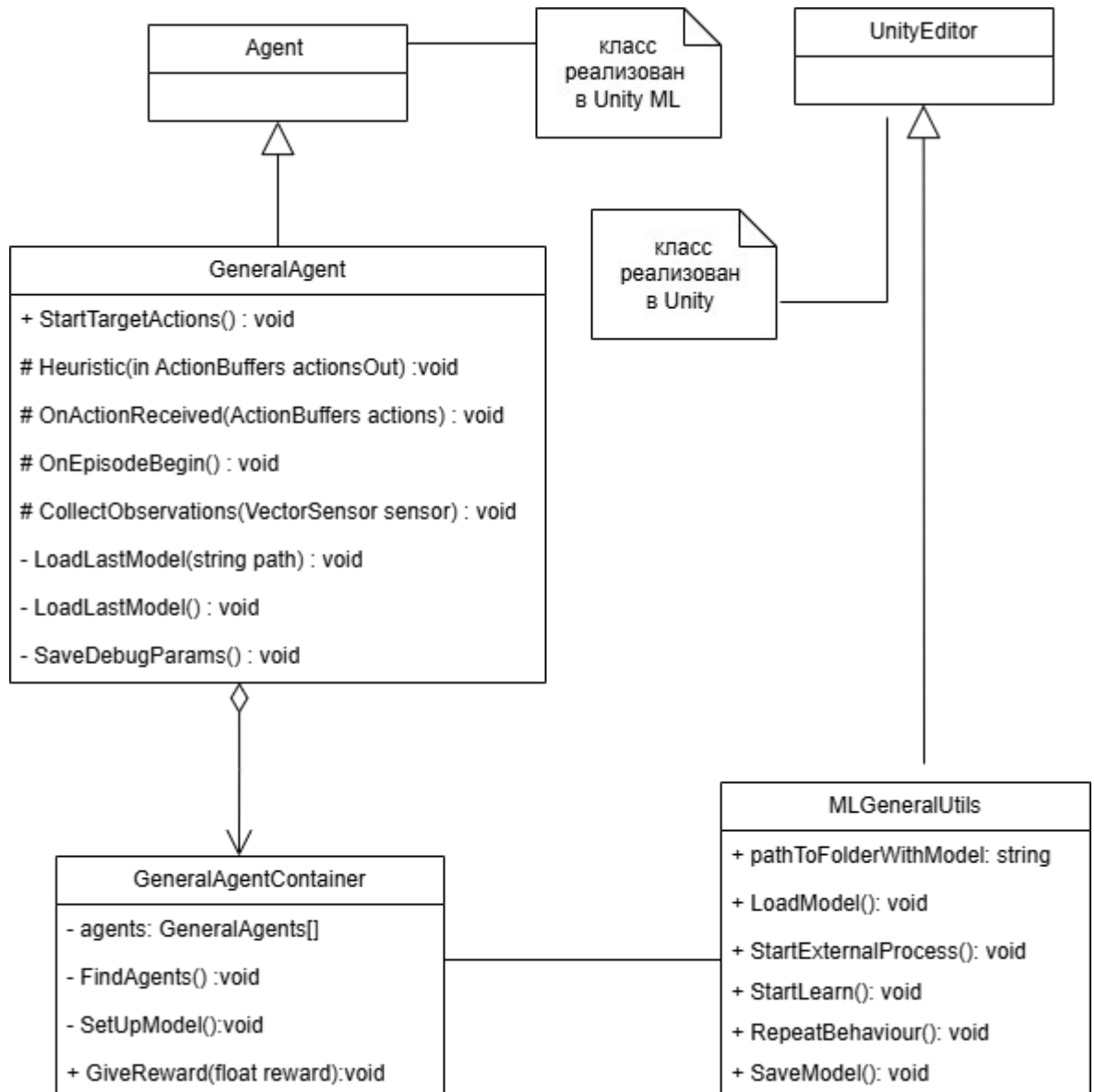


Рисунок 7 – Диаграмма классов

Краткое описание классов представлено в таблице 4.

Таблица 4 – Краткое описание классов

| Класс                 | Описание                                                                                                                                        |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Agent                 | Базовый класс для любого агента. Поставляется вместе с Unity ML и служит провайдером к консольной утилите.                                      |
| GeneralAgent          | Класс наследуемый от Agent и расширяющий его возможности для обучения агентов.                                                                  |
| GeneralAgentContainer | Класс-контейнер для GeneralAgent. Используется для обучения агентов групповому поведению и отладки этого поведения.                             |
| MLGeneralUtils        | Утилит-класс, наследуемый от UnityEditor. Обеспечивает взаимодействие с файлами отладки, обученными моделями и менеджментом консольной утилиты. |
| UnityEditor           | Базовый класс для написания расширений для Unity.                                                                                               |

Для описанных классов системы в таблице 5 представлена спецификация методов.

Таблица 5 – Описания основных методов системы

| Название  | Назначение                                                                    | Входные параметры                            |
|-----------|-------------------------------------------------------------------------------|----------------------------------------------|
| Heuristic | Моделирование поведения пользовательскими действиями.<br>Сохранение действий. | ActionBuffer - набор из изменений параметров |

Продолжение таблицы 5

|                     |                                                                                                                |                                                                                                     |
|---------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| OnActionReceived    | Применение параметров выданных нейронной сетью или методом Neuristic. Сохранение этих действий в временной ряд | ActionBuffer - набор из изменений параметров                                                        |
| OnEpisodeBegin      | Начало эпизода обучение и сброс предыдущих параметров                                                          | –                                                                                                   |
| CollectObservations | Отправка данных сенсоров для нейронной сети                                                                    | VectorSensor – коллекция, в которую прикладной код проекта добавляет способы взаимодействия с миром |

Все описанные методы позволяют осуществлять обучение и отладку поведения агентов. Стоит отметить, что метод CollectObservations принимает объект класса VectorSensor, в который с помощью метода AddObservation добавляется наблюдение. Наблюдение может представлять собой вектор, число или логическое значение. Перед передачей этих данных в API ML-Agents они автоматически нормализуются, чтобы предотвратить нестабильное обучение, резкие изменения градиента и переобучение на признаки, имеющие большие веса.

Метод Neuristic обеспечивает ввод данных для действий агента. Может производиться как с клавиатуры так и с любого другого управляющего устройства. Кроме того метод позволяет сохранить временной ряд пользовательских действий и взаимодействий с приложением для предобучения агента на основании их.

Процесс обучения в одной обучающей среде представлен на диаграмме деятельности (рисунок 8).

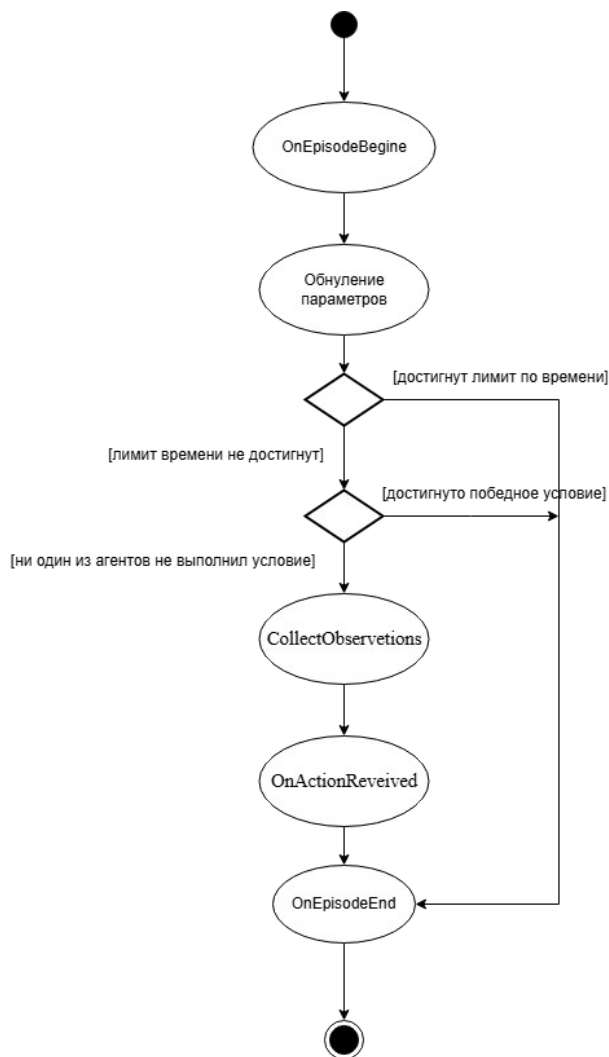


Рисунок 8 – Диаграмма деятельности

Диаграмма деятельности описывает алгоритм обучения в одной среде. При этом возможно одновременное функционирование нескольких таких сред. В начале обучения для каждого агента вызывается метод `OnEpisodeBegin`, который сбрасывает предыдущие награды агента. Затем выполняется сброс всех внутренних состояний пользовательского агента.

Далее алгоритм переходит в бесконечный цикл, в котором отслеживается предельное время выполнения задачи, поставленной перед агентом. При достижении предельного времени или выполнении условия завершения игры эпизод завершается, награды подсчитываются, и в параметры нейронной сети вносятся корректировки.

Длительность цикла и его запуск могут быть сконфигурированы. При каждом запуске цикла вызывается метод `CollectionObservation`. После сбора

всех доступных данных с сенсоров и их передачи в нейронную сеть она выполняет предсказание значений и передает объект `ActionBuffer` в метод `OnActionReceived`. Этот метод преобразует числовые параметры и обеспечивает выполнение необходимых действий пользовательским кодом.

## 4.2 Анализ доступных ML алгоритмов

Для быстрого обучения в Unity ML используется обучение с подкреплением. Существует несколько реализаций алгоритмов использующих Actor-Critic-подход в UnityML:

- политика проксимальной оптимизации (Proximal Policy Optimization, PPO),
- мягкая актор-критик (Soft Actor-Critic, SAC),
- мультиагентный мягкий актор-критик (Multi-Agent Soft Actor-Critic, MA-SAC).

Принцип выбора алгоритм представлен в таблице 6.

Таблица 6 – Принцип выбора алгоритма

| Тип задачи                       | Алгоритм | Критерий                                                              | Пример использования                                |
|----------------------------------|----------|-----------------------------------------------------------------------|-----------------------------------------------------|
| Один агент, дискретные действия  | PPO      | Простые задачи где важна стабильность обучения                        | Шахматы, китайская игра GO                          |
| Один агент, непрерывные действия | SAC      | Сложные задачи с непрерывными действиями                              | Обучение роботов на производствах, симуляция полета |
| Один агент, сложная среда        | SAC      | Для обучения в сложной среде, при этом используется накопление данных | Манипулятор с камерой                               |

## Продолжение таблицы 6

|                                         |        |                                            |                     |
|-----------------------------------------|--------|--------------------------------------------|---------------------|
| Много агентов и требуется кооперация    | MA-SAC | Когда агенты должны помогать друг другу    | Командный футбол    |
| Много агентов, кооперация и конкуренция | MA-SAC | Одни агенты в команде, другие – противники | Захват флага, RTS   |
| Ограниченные ресурсы, простая среда     | SAC    | Обучение должно быть быстрым и простым     | Казульные мини-игры |

Реализация этих алгоритмов охватывает большинство задач связанных с обучением ИИ-агентов. Алгоритмы демонстрирует устойчивость и применимость в сложных системах. Большинство игр и взаимодействий может быть реализовано с их использованием. Применимость для моделирования подтверждают исследования университета Беркли и Unity Technologies [20, 21].

Эти алгоритмы уже реализованы в Python API Unity ML-Agent и доступны в качестве готовых опций. Для модификации алгоритмов или введение дополнительных гиперпараметров для обучения в разработанной системе представлено отдельное расширение, позволяющее создавать Python-скрипты реализующие собственные RL-алгоритмы и использовать их при обучении.

### 4.3 Требования к прототипу для тестирования системы

Для сравнения эффективности нейронных сетей и деревьев поведения при проектировании искусственного интеллекта, имитирующего тактику противника, будет проведено тестирование в игровой среде «Захват флага» со следующими правилами: игрок должен захватить флаг и доставить его на свою базу, при этом атака противника разрешена исключительно при отсутствии флага. Уничтожение оппонента приносит дополнительные очки.

Данный вид игры был выбран, поскольку он позволяет оценить наиболее сложный сценарий обучения нейронных сетей, а именно одновременное использование кооперации и противодействия противнику.

Структура дерева поведения для данной задачи представлена на рисунке 9.

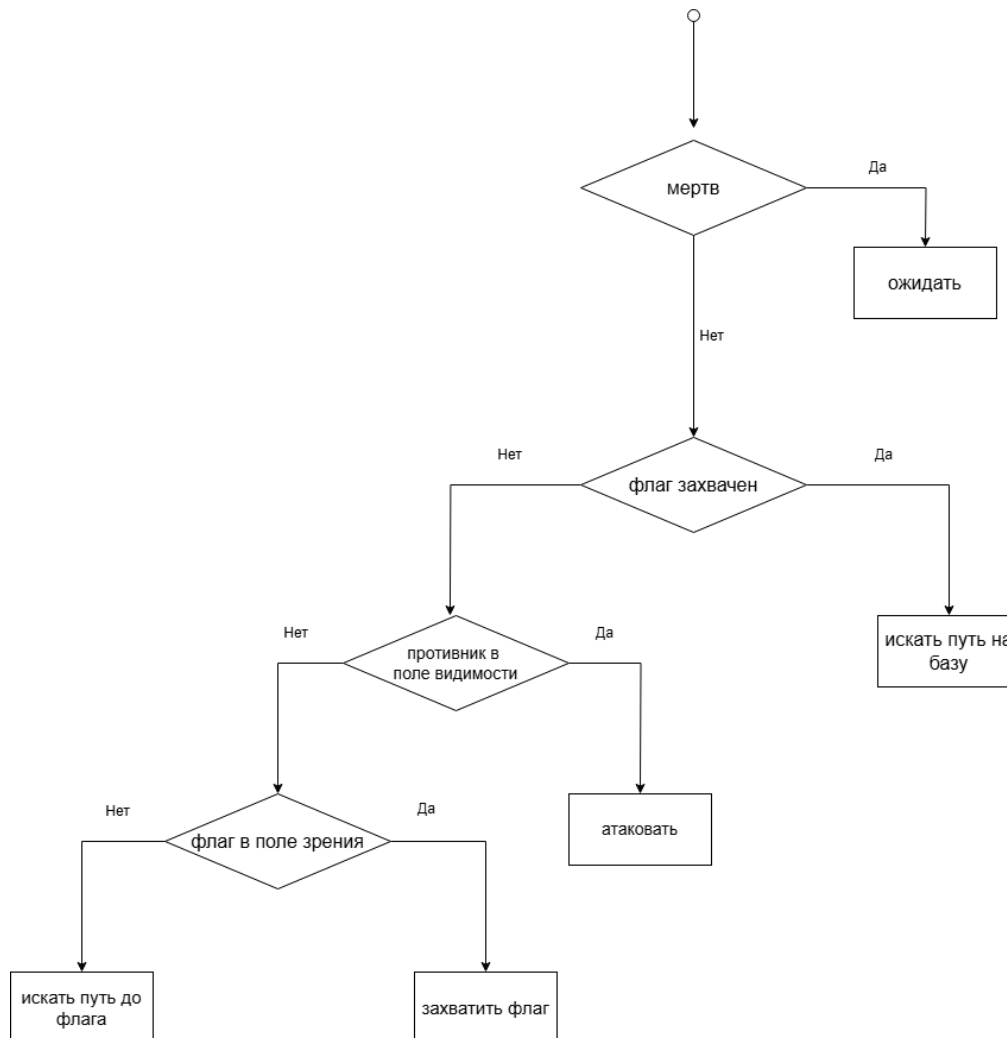


Рисунок 9 – Дерево поведения

Для нейронной сети можно определить следующие атомарные действия:

- атаковать,
- двигаться,
- захватить флаг,
- умереть.

Комбинации всех действий полностью покрывают дерево решений, а именно:

- атаковать,
- искать путь на базу,
- искать путь до флага,
- убегать от игрока,
- захватить флаг,
- ожидать.

Данный набор атомарных действий позволит полноценно протестировать полноту и глубину поведения как дерева поведения, так и нейронной сети.

#### **4.4 Обзор возможностей обучения через расширение**

Для корректировки существующий набор гиперпараметров можно редактировать через интерфейс Unity.

Интерфейс разработан на основе базового класса UnityEditor и функционирует исключительно в режиме редактора. Обучение агентов осуществляется только в среде UnityEditor, поэтому данная часть кода не включается в разрабатываемое приложение, как и все зависимости в виде Python-модулей.

Расширение можно открыть через окно редактора вкладки «Tools», интерфейс системы представлен на рисунке 10. Расширение представляет из себя динамическое окно, которое можно разместить в вкладки редактора. Графики и сводная информация о проведенном обучении появляется сразу после его завершения, при условии того, что Unity Editor еще открыт и Python-среда доступна.

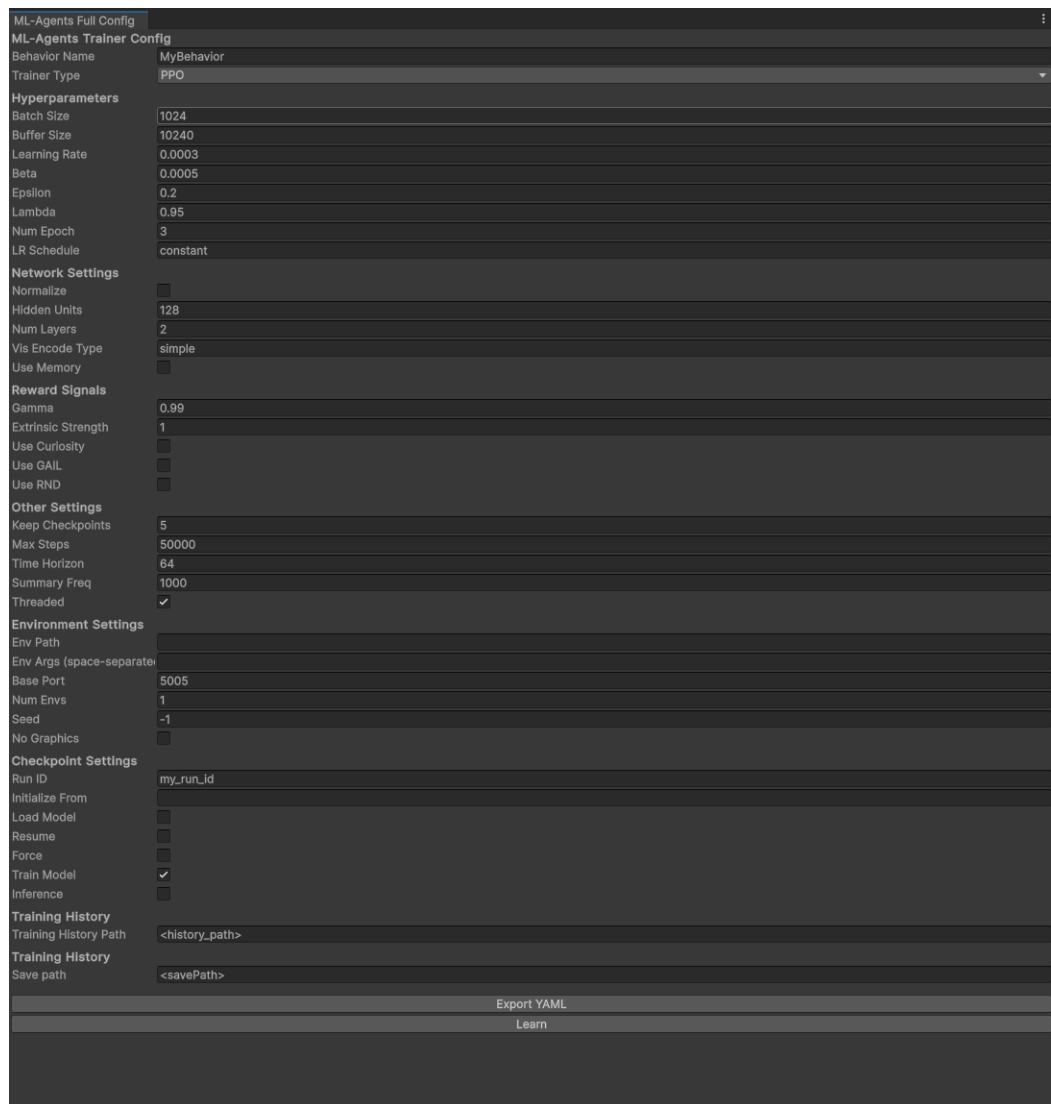


Рисунок 10 – Интерфейс расширения.

Расширение обеспечивает редактирование гиперпараметров с последующим сохранением в файлы формата YAML, а также позволяет задавать пути для сохранения конфигураций и журналов обучения. При активации функции Learn, Unity система автоматически генерирует Standalone-сборку приложения и инициирует запуск изолированных сред в количестве, указанном в аргументах окружения (Env Args). Каждая среда представляет собой независимое Unity-приложение, функционирующее в режиме без графического интерфейса и с выделенным адресным пространством (приложение A.1). Основные гиперпараметры, используемые в процессе обучения, приведены в таблице 7.

Таблица 7 – Описание основных гиперпараметров

| Параметр             | Описание                                                                                                         | Пример                         |
|----------------------|------------------------------------------------------------------------------------------------------------------|--------------------------------|
| batchSize            | Количество экспериментов в каждой итерации градиентного спуска.                                                  | PPO: 512-5120; SAC: 128-1024   |
| bufferSize           | Для PPO – количество опыта перед обновлением модели. Для SAC – максимальный размер буфера.                       | PPO: 2048-409600; SAC: 5e4-1e6 |
| learningRate         | Начальная скорость обучения для градиентного спуска.                                                             | 1e-5 – 1e-3                    |
| beta                 | Сила энтропийной регуляризации, увеличивает случайность действий для исследования.                               | 1e-4 – 1e-2                    |
| epsilon              | Порог расхождения между старой и новой политикой PPO. Меньшее значение стабилизируют обучение, но его замедляет. | 0.1 – 0.3                      |
| lambda               | Параметр регуляризации для PPO. Определяет баланс между смещением и дисперсией при оценке.                       | –                              |
| trainerType          | Тип тренера: ppo, sac или rosa                                                                                   | Ppo, sac, rosa                 |
| numEpoch             | Количество проходов через буфер опыта при оптимизации                                                            | 3 – 10                         |
| learningRateSchedule | Расписание изменения скорости обучения. linear (снижается линейно) для PPO, constant (постоянная) для SAC.       | linear или constant            |
| normalize            | Нормализация векторных наблюдений. Полезна для сложных задач с непрерывными действиями.                          | true/false                     |

### Продолжение таблицы 7

|                   |                                                                                                                                                      |                            |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| hiddenUnits       | Количество нейронов в скрытых слоях нейронные сети.<br>Больше для сложных задач, меньше для простых.                                                 | 32 – 512                   |
| numLayers         | Количество скрытых слоев в нейронные сети. Больше слоев — сложнее модель                                                                             | 1 – 3                      |
| useMemory         | Активация памяти (LSTM). Используется для задач, требующих запоминания состояния.                                                                    | –                          |
| memorySize        | Размер памяти LSTM. Должен быть кратен 2.                                                                                                            | 32 – 256                   |
| sequenceLength    | Длина последовательности для обучения LSTM. Увеличивает способность запоминать, но замедляет обучение.                                               | 4 – 128                    |
| useGAIL           | Активация GAIL (имитационное обучение). Требуется демонстраций.                                                                                      | –                          |
| curiosityStrength | Сила вознаграждения за любопытство (Curiosity).<br>Управляет влиянием внутренней мотивации. Слишком высокое значение может подавить внешние награды. | 0.001 – 0.1<br>(Curiosity) |

Это основные параметры используемые для корректировки обучения. Полный список параметров, которые влияют на обучение можно изменить в yaml-файле [22].

### 4.5 Оценка качества обучения агентов

Для оценки качества обучения агентов из расширения можно открыть TensorBoard. В доске можно отследить качество обучения. Качество обучения можно оценить несколькими объективными параметрами. Для оценки результатов стоит обратить внимание на несколько самых показательных метрик указанных в таблице 8.

Таблица 8 – Метрики оценки качества обучения

| Метрика           | Описание                                                              | Интерпритация                                                                                            |
|-------------------|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| Cumulative Reward | Суммарное вознаграждение за эпизод.                                   | Стабильный рост – агент учиться получать больше награды. Падение или колебания – переобочение.           |
| Policy Loss       | Ошибка обновления политики (разница между старой и новой стратегией). | Высокие значения – нестабильность в обучении. Снижение – политика стабилизируется.                       |
| Value Loss        | Ошибка предсказания ценности состояний.                               | Высокие колебания – плохая оценка будущих наград. Стабильное снижение – ценность предсказывается точнее. |
| Entropy           | Степень случайности действий агента.                                  | Стабильное снижение – обучение стабилизируется                                                           |
| Эло               | Показывает степень обучения агента                                    | Чем выше Эло тем более адекватно действует агент.                                                        |

Одним из важнейших показателей является Эло (Elo) – это система, созданная Арпадо Эло для оценки силы игроков в соревновательных играх, например, в шахматах. В обучение с подкреплением оно используется для отслеживания прогресса агентов при их игре друг против друга. Формула расчета Эло рейтинга производится по формуле.

$$\Delta ELO = K * (S - E),$$

где  $\Delta ELO$  – изменение рейтинга;

$K$  – коэффициент чувствительности;

$S$  – фактический результат;

$E$  – ожидаемый результат.

Ожидаемый результат  $E$  вычисляется по формуле :

$$E = \frac{1}{1 + 10^{\frac{R_{opp} - R_{agent}}{400}}} \quad (1)$$

где  $R_{opp}$  – Эло рейтинг оппонента;

$R_{agent}$  – текущее Эло рейтинг агента.

Эло помогает объективно оценить силу агента в динамически и быстро меняющейся среде, чем выше Эло тем лучше агент справляется с противниками [23].

Для обучения с подкреплением используется показатель Эло, а не коэффициент побед, поскольку вычисление коэффициента побед по формуле 2 требует одинакового количества матчей против одного и того же противника.

$$WinRate = \frac{W}{K} \quad (2)$$

где  $W$  – количество побед;

$K$  – общее число сыгранных игр.

В отличие от коэффициента побед, рейтинг Эло не требует фиксированного противника для оценки качества обучения, что позволяет менять агентов и, тем самым снижая, вероятность переобучения.

## **5 Тестирование системы**

### **5.1 Разработка тестовой среды**

Для верификации работоспособности системы необходимо разработать игровой прототип «Захват флага». Данный тип игры выбран в качестве тестового стенда, поскольку включает ключевые элементы для комплексной проверки системы:

- взаимодействие между союзными агентами (коллаборацию),
- сложную многоэтапную задачу,
- противоборствующую команду с аналогичными целями.

На первом этапе требуется определить набор атомарных непрерывных действий. Для реализации в C#-скрипте предлагаются следующие базовые действия агента:

- движение,
- поворот,
- атака,
- захват флага.

Для обеспечения согласованных действий агентов, они должны быть организованы в группы. В разработанной системе используется механизм группового вознаграждения. Победа в раунде приводит к распределению награды между всеми агентами в группе-победителя, в то время как проигравшая команда получает соответствующий штраф. Код управления и организацией среды представлен в Приложении А.2.

Подход позволяет эффективно обучать группы агентов выполнению базовых задач в простых средах при большом количестве итераций. Оптимизация параметров обучения дает возможность сократить количество повторения

Для работы в сложных средах предусмотрена система пропорционального вознаграждения, где простые действия агентов

поощряются соразмерными наградами. Схема вознаграждений за различные действия представлена в таблице 9.

Таблица 9 – Вознаграждение агентов за действия

| Действие                       | Мотивация                                                                                    | Вознаграждение |
|--------------------------------|----------------------------------------------------------------------------------------------|----------------|
| Захват флага                   | Награда обеспечивает заинтересованность агентов в поиске флага.                              | [0.4; 0.6]     |
| Агент движется с флагом к базе | Награда заставляет агента не стоять на месте и мотивирует его двигаться в сторону своей базы | [0.09; 0.1]    |
| Убийство противника            | Награда мотивирует агента совершать убийства противников                                     | [0.9; 1]       |
| Соприкосновение с союзником    | Штраф мотивирует агента не сталкиваться с союзниками                                         | [-0.2, 0.09]   |
| Смерть                         | Штраф мотивирует агента избегать прямых атак                                                 | [-0.5, -0.3]   |
| Победа в раунде                | Мотивирует агентов выполнить главную цель игры.                                              | [1;5]          |

Дополнительные вознаграждения способствуют сглаживанию кривой обучения. Награда за победу в раунде нормируется на время, прошедшее с начала обучения, что мотивирует агентов минимизировать продолжительность игры. При этом величина штрафа за поражение специально ограничена, чтобы сохранить возможность рискованных действий.

Важно отметить, что размер вознаграждения случайным образом выбирается из заданного диапазона, что предотвращает переобучение модели и фиксацию агента на ограниченном наборе стратегий. Код агента представлен в Приложении А3.

Обучение агентов разделяется на эпизоды. Каждый эпизод может завершиться одним из трех состояний:

- победа первой команды,

- победа второй команды,
- истечение отведенного времени эпизода.

По окончании эпизода среда возвращается в исходное состояние. Стартовые позиции агентов незначительно изменяются относительно первоначальных, что предотвращает запоминание агентами фиксированных стратегий. Позиция флага также подвергается незначительному смещению.

Для обеспечения возможности ориентации агента в сложной среде был интегрирован компонент `RayPerceptionSensor`, предоставляющий агенту информацию об окружающей среде посредством проецирования физических лучей. Визуализация представлена на рисунке 11.

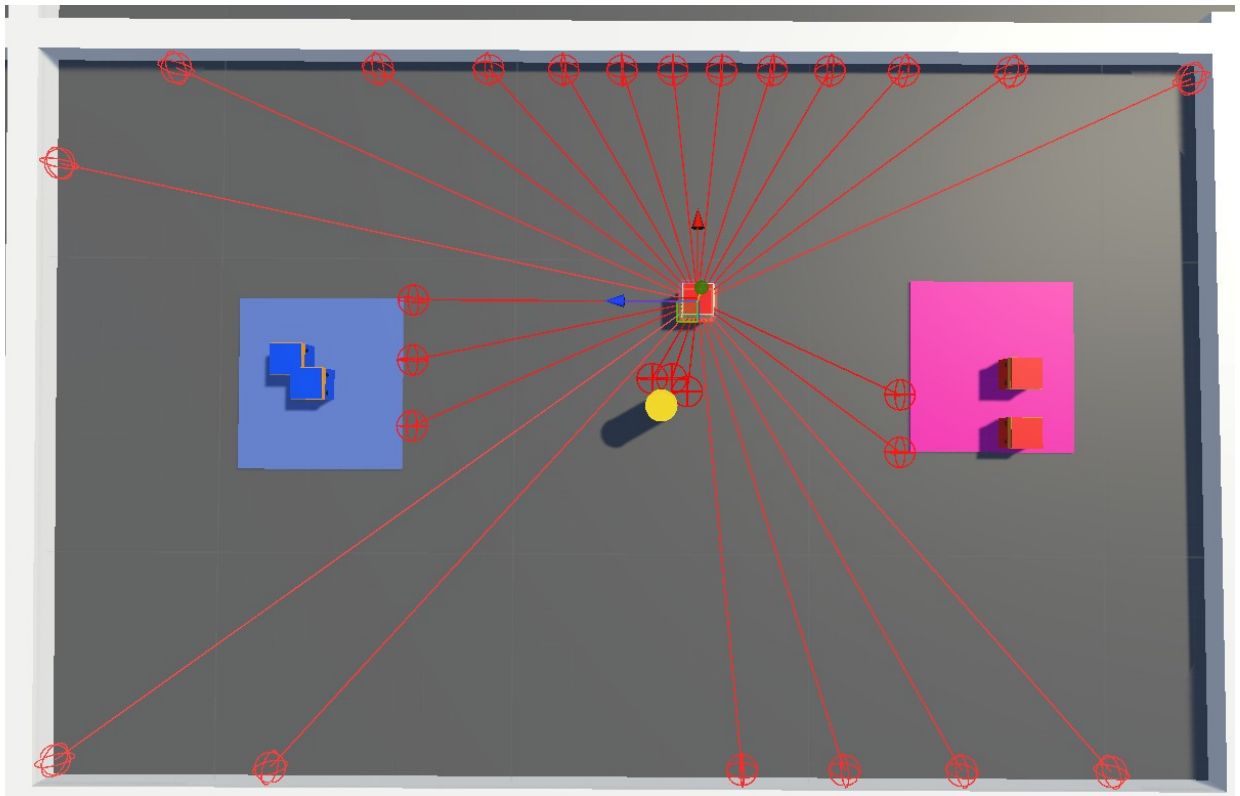


Рисунок 11 – Визуализация работы `RayPerceptionSensor`

Компонент `RayPerceptionSensor` формирует 13 лучей, исходящих из центра агента, с максимальным углом охвата 180 градусов. Такой набор параметров обеспечивает высокое качество ориентации агентов в среде при невысоких затратах на поиск препятствий. Каждый луч представляет из себя вызов системной функции `Unity Raycast`, которая проецирует луч на

указанные физические слои. При этом лучи также проецируются на «невидимые стены», которые обозначают место расположения исходной точки команды.

Для обеспечения адаптивности агентов к различным условиям были подготовлены три варианта обучающих сред:

- свободное пространство без препятствий,
- лабиринт,
- процедурно генерируемая карта.

Визуально среды с пустым пространством представлена на рисунке 12.

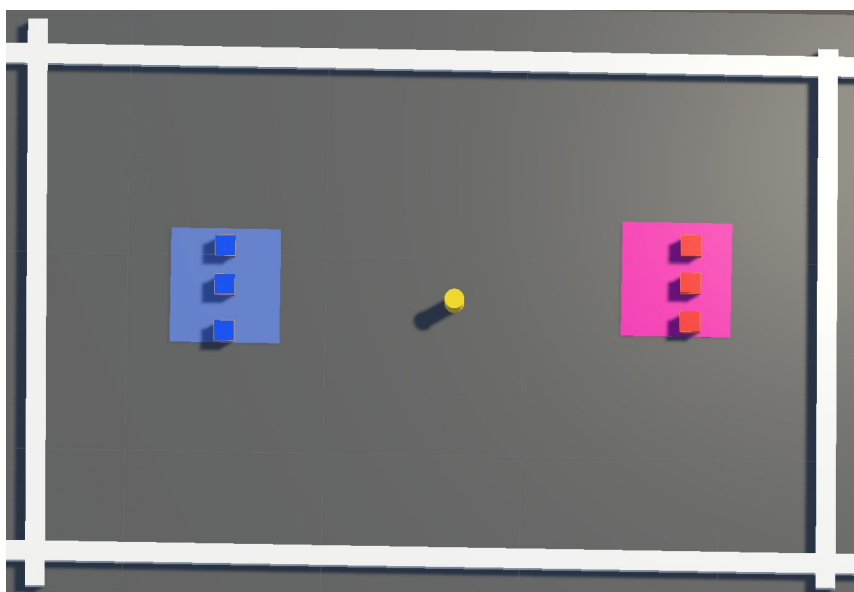


Рисунок 12 – Пустое пространство без препятствий

Данная среда использовалась на первом этапе обучения нейронной сети для формирования у агентов навыков ориентации в пространстве и выполнения основных действий, таких как: атака оппонента, перемещение флага, движение к базе, избегание урона от оппонента, сотрудничество между агентами. Эти поведенческие стратегии легли в основу более сложного взаимодействия агентов, которые обучались в условиях большей неопределенности и большого количества стохастических факторов.

В редакторе Unity с использованием примитивных объектов, поставляемых вместе с редактором, была создана усложненная среда для обучения. Ее визуальное представление изображено на рисунке 13.

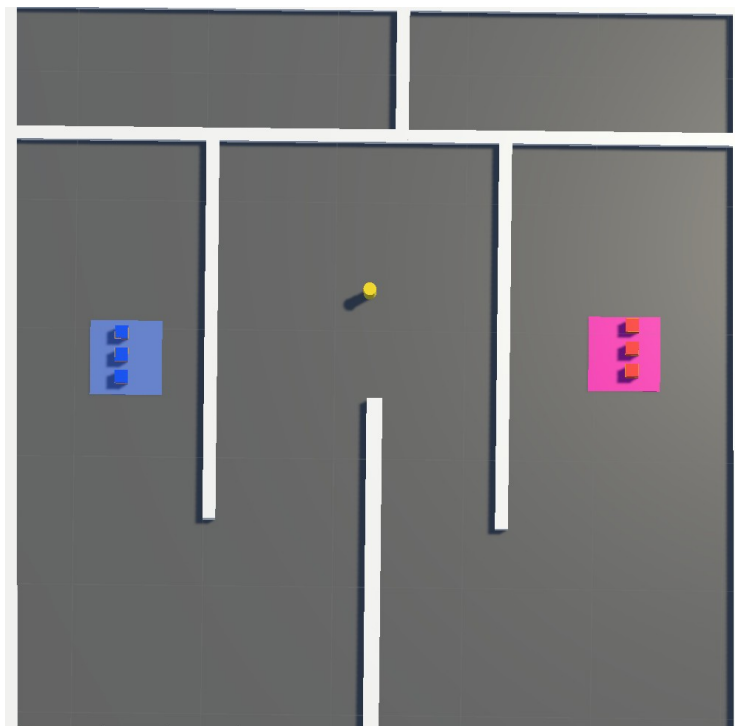


Рисунок 13 – Среда лабиринт

Среда в виде лабиринта использовалась на втором этапе обучения для формирования у агентов навыков избегания препятствий и поиска оптимальных путей к целевому объекту и обратно. Данный вид среды также позволяет снизить вероятность переобучения агентов и заучивания маршрута к флагу. На этот этап потребовалось в два раза больше шагов, чем на первый, что связано с усложнением игровой среды и необходимостью устранения эффекта заучивания, сформированного на первом этапе.

Разнообразные среды обучения являются важным этапом при обучении агентов, поэтому следует использовать либо множество разных обучающих сред, либо процедурно генерируемые уровни.

Для обучения агентов был применен подход процедурной генерации среды обучения. Пример процедурно сгенерированной среды представлен на рисунке 14.

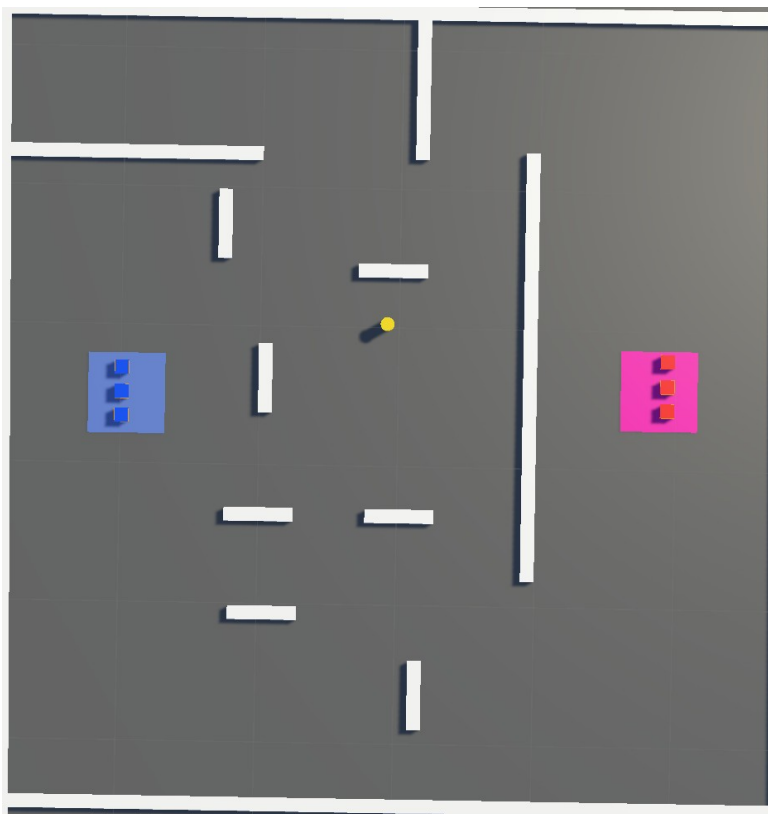


Рисунок 14 – Среда с случайными препятствиями

Последней средой является среда с процедурно генерируемыми препятствиями и случайными смещениями агентов и флагов. Данный подход позволяет устранить заучивание агентами выигрышных стратегий и особенностей одной среды.

Данный набор сред обучения дает возможность обучаться агентам в разных условиях и избежать переобучения.

## **5.2 Обучение агентов в тестовой среде**

Обучение проводилось на ЭВМ, включающем в себя видеокарту Nvidia RTX 4060 Ti и процессор AMD Ryzen 5 7500F (6 ядер).

Было запущено 8 параллельных Unity-приложений, в каждом из которых создано 1240 рабочих окружений для обучения агентов. Количество обучающих сред и запущенных приложений ограничивалось вычислительной мощностью процессора.

Физические расчеты (преобразования, движение и коллизии) в Unity выполнялись процессором, тогда как обучение нейронной сети осуществлялось с использованием технологии CUDA без полной загрузки вычислительных ядер.

Для того, чтобы нейронная сеть смогла выполнить задачи с заданной точностью, потребовалось 6 000 000 шагов обучения, что заняло 7200 секунд (2 часа). Скорость обучения (learning\_rate) была установлена на уровне 0.001 для обеспечения необходимого качества обучения. Полная конфигурация параметров обучения представлена в листинге А.3.

### 5.3 Результаты обучения

Для оценки результатов обучения была использована возможность разработанной системы по сохранению и анализу результатов обучения через TensorBoard. Оценка Эло обучения агентов равна 4128, что при исходном 1200, показывает значительное увеличение способности агентов побеждать. На рисунке 12 представлен график увеличения Эло агентов.

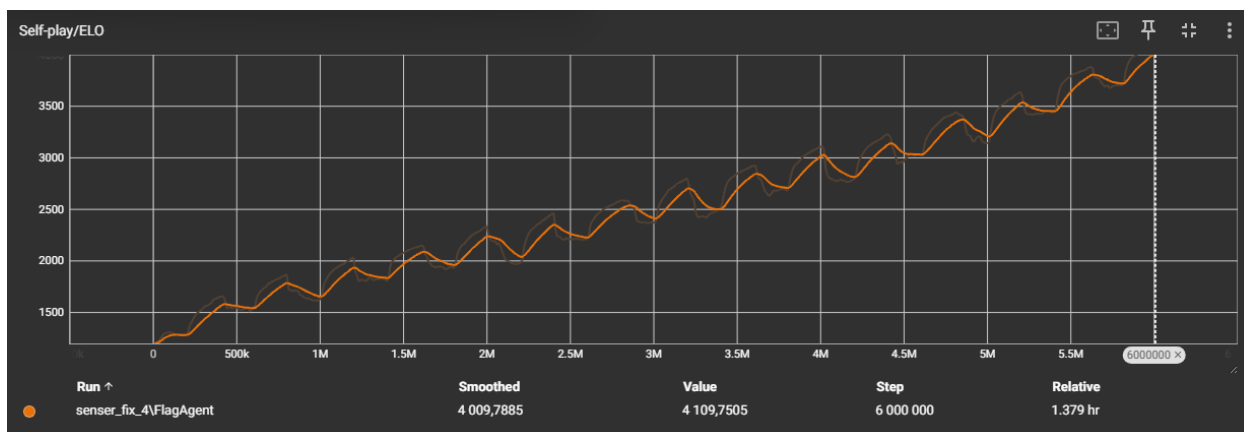


Рисунок 15 – График увеличения Эло агентов.

Скачки и падения, можно объяснить периодом смены стратегии агентом. Также на небольшие скачки Эло может влиять процедурно сгенерированные среды. Также политик PPO намеренно добавляет шум, для того, чтобы агент мог исследовать новые среды.

На рисунке 13 представлен результат Policy/Entropy, говорящее о степени случайности действий.

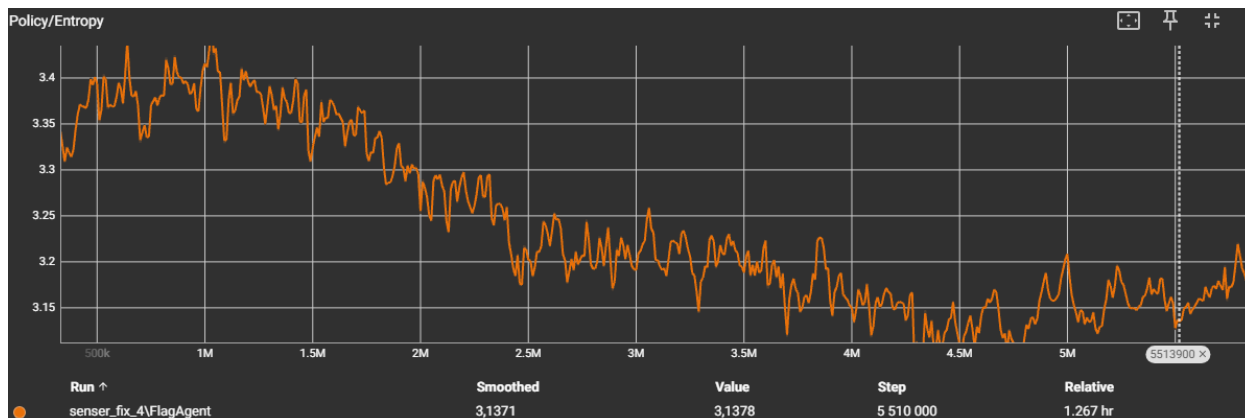


Рисунок 16 – График уменьшения Policy/Entropy

На графики заметно планомерное уменьшение случайности действий агента, что говорит о качестве обучения.

На рисунке 14 представлена динамика отношения Losses/Baseline Loss в течение первых 1 100 000 шагов обучения.

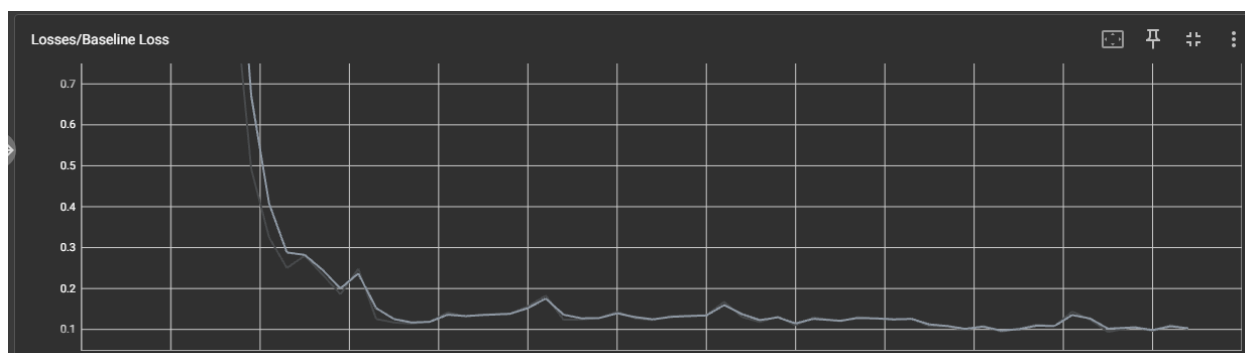


Рисунок 17 – График уменьшения функции потерь

Анализ графика показывает, что агент достигает значения 0.1 уже на первом миллионе шагов. К 6 млн. шагу показатель снижается до 0.001, что свидетельствует о значительном улучшении качества обучения.

На рисунке 15 представлен график зависимости функции потерь от стоимости (loss-to-value), демонстрирующий степень расхождения между ожидаемой и фактической оценки выгодности текущих состояний.

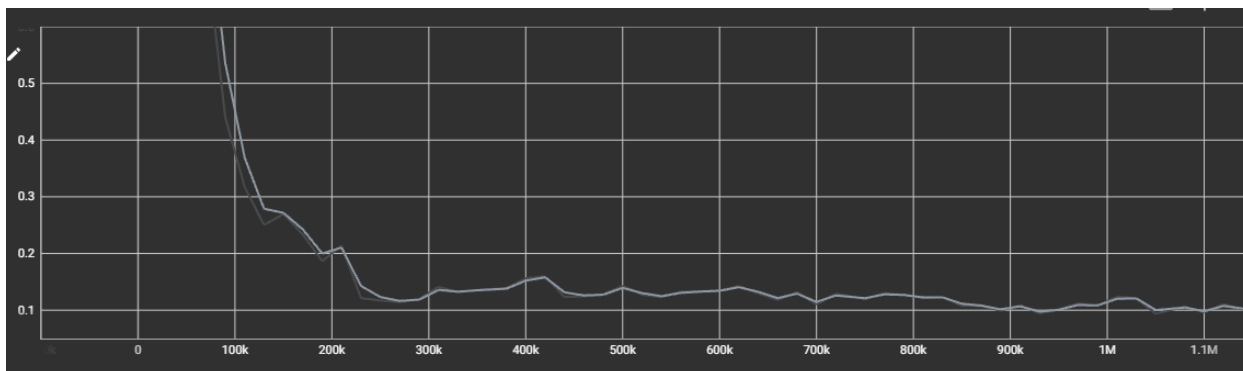


Рисунок 18 – график зависимости потерь от стоимости

Анализ графика демонстрирует, что агент достигает значения 0.9 к 1 000 000 шагу обучения. В ходе полного цикла обучения показатель снижается до 0.05, что подтверждает эффективность процесса обучения.

Результаты обучения на 6 000 000 шаге можно посмотреть в таблице 10.

Таблица 10 – Результаты обучения

| Метрика           | Значение | Интерпретация                                                                    |
|-------------------|----------|----------------------------------------------------------------------------------|
| Cumulative Reward | 6.05     | Высокая награда относительно максимальный.                                       |
| Policy Loss       | 0.05     | Ценность предсказаний высокое                                                    |
| Value Loss        | 0.03     | Ценность предсказаний высокое                                                    |
| Entropy           | 0.38     | Обучение стабилизировалось                                                       |
| Эло               | 4102     | Увеличение параметра свидетельствует о увеличении адекватности поведения агента. |

На основе представленных метрик можно сделать вывод, что показатель Эло, изначально составлявший 1200, увеличился почти в два раза, что свидетельствует о повышении адекватности поведения агентов. При этом точность предсказания поведения остается высокой, а процесс обучения стабилизировался.

Качество обучения также можно измерить эмпирически через параметры представленные в таблице 11.

Таблица 11 – Эмпирическая оценка

| Эмпирический критерий                          | Шаг обучения |             |           |           |           |
|------------------------------------------------|--------------|-------------|-----------|-----------|-----------|
|                                                | 100<br>тыс.  | 500<br>тыс. | 2<br>млн. | 4<br>млн. | 6<br>млн. |
| Количество завершившихся раундов в процентах   | 32 %         | 67%         | 85%       | 90%       | 98%       |
| Количество устраненных противников в процентах | 42%          | 49%         | 70%       | 75%       | 85%       |
| Среднее время раунда в секундах                | 289          | 230         | 120       | 53        | 20        |

Эмпирические критерии были выбраны таким образом, чтобы продемонстрировать эффективность обучения. Количество завершенных раундов представляет собой процент от общего числа запущенных раундов, которые завершились либо доставкой флага на базу, либо устранением всех противников. Наблюдается значительный рост количества завершенных раундов, что свидетельствует о корректности проведенного обучения.

Количество устраненных противников в раунде увеличилось до 85%, что связано с тем, что в некоторых случаях агенты либо не сталкивались с противниками, либо успевали захватить флаг и доставить его на базу, тем самым завершая раунд.

Одним из ключевых показателей является длительность раунда. Быстрое завершение раундов указывает на то, что агенты искусственного интеллекта успешно ориентируются в среде и достигают поставленной цели.

Обученная нейронная сеть демонстрирует высокую эффективность при выполнении задач и завершении раундов с максимально возможным количеством очков.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения магистерской диссертации был осуществлен обзор предметной области, в процессе которого была рассмотрена актуальность создания системы позволяющей применять нейронные сети для моделирования поведения противников. Был проведен обзор существующих программных средств, специализирующихся на внедрение нейронных сетей в игры. В рамках работы была поставлена задача на разработку проекта. В процессе проектирования приложения была проведена разработка требований к системе и сформулированы параметры ее тестирования. Процесс разработки требований включал в себя определенные требования к архитектуре и используемым инструментальным средствам. Было разработано расширение для Unity интегрированное с Unity ML и TensorFlow. Для тестирования системы была обучена нейронная сеть позволяющая моделировать поведение противников в игре «Захват флага».

В результате выполнения работы была разработана система позволяющая обучать нейронные сети для моделирования поведения противников в играх, а также прототип проекта, демонстрирующий функционал системы.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. World Economic Forum. "Gaming has thrived during pandemic lockdowns, and PwC predicts growth." [<https://www.weforum.org/agenda/2022/07/gaming-pandemic-lockdowns-pwc-growth/>] Опубликовано 1 июля 2022 года. Дата обращения: 24 декабря 2023 года.
2. No Man's Sky. - [<https://www.nomanssky.com/>] (ссылка на интернет-ресурс). Дата обращения: 24 декабря 2023 года.
3. Bethesda. Starfield. - [<https://bethesda.net/en/game/starfield>] (ссылка на интернет-ресурс). Дата обращения: 24 декабря 2023 года.
4. NVIDIA. Redefines Game AI With ACE Autonomous Game Characters. - [<https://www.nvidia.com/en-us/geforce/news/nvidia-ace-autonomous-ai-companions-pubg-naraka-bladepoint/>] (ссылка на интернет-ресурс). Дата обращения: 29 декабря 2024 года.
5. Millington, I. AI for Games. – 3rd ed. – CRC Press, 2019. – 958 p. – ISBN 978-1138483972 .
6. Сильвестер, Т. (Tynan Sylvester). Designing Games: A Guide to Engineering Experiences. – O'Reilly Media: Sebastopol, CA, 2013. – 300 с. – ISBN 978-1-4493-0607-6.
7. Баррера, Р. (Ray Barrera). Unity AI Game Programming. - СПб: БХВ-Петербург, 2018. - 352 с. - ISBN 978-5-9775-4225-7.
8. Бург, Д. М. (David M. Bourg). AI For Game Developers. - СПб: БХВ-Петербург, 2004. - 384 с. - ISBN 5-94157-170-3.
9. Liu, J., Xiao, M., Wen, J., Kang, J., Zhang, R., Zhang, T., Niyato, D., Zhang, W., Liu, Y. The Ethics of AI in Games // IEEE Transactions on Cognitive Communications and Networking. – 2025. – 14 января. – DOI: 10.1109/TCCN.2025.3529747. – Electronic ISSN 1949-3045, CD ISSN 2371-9850.

10. Unreal Engine документация. URL: <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation>. Дата обращения: 5 июня 2024 года.
11. Unity документация. URL: <https://docs.unity.com/>. Дата обращения: 5 июня 2024 года.
12. Godot документация. URL: <https://docs.godotengine.org/ru/4.x/index.html>. Дата обращения: 5 июня 2024 года.
13. Microsoft Visual studio for unity. URL: <https://learn.microsoft.com/ru-ru/visualstudio/gamedev/unity/get-started/getting-started-with-visual-studio-tools-for-unity?pivot=windows>. Дата обращения: 5 июня 2024 года.
14. Rider for Unity. URL: <https://www.jetbrains.com/idea/dotnet-unity/> Дата обращения: 29 декабря 2024 года.
15. MonoDeveloper IDE. URL: <https://www.monodevelop.com/documentation/> (дата обращения 15.12.2024).
16. Единая информационная система в сфере закупок Российской Федерации. – [<https://zakupki.gov.ru/epz/order/notice/notice223/common-info.html?noticeInfoId=15921013>] (ссылка на интернет-ресурс). Дата обращения: 5 июня 2024 года.
17. Unity Technologies. ANN Perceptron. – [<https://assetstore.unity.com/packages/tools/behavior-ai/ann-perceptron-117766>] (ссылка на интернет-ресурс). Дата обращения: 20 апреля 2024 года.
18. Unity Technologies. Simple Neural Network. – [<https://assetstore.unity.com/packages/tools/ai-ml-integration/simple-neural-network-281473>] (ссылка на интернет-ресурс). Дата обращения: 20 апреля 2024 года.
19. Unity ML Agent Overview. URL: <https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html>
20. Cohen, A., Teng, E., Berges, V.-P., Dong, R.-P., Henry, H., Mattar, M., Zook, A., Ganguly, S. On the Use and Misuse of Absorbing States in Multi-

agent Reinforcement Learning. – arXiv:2111.05992v2 [cs.LG]. – 7 июня 2022 года. – [<https://arxiv.org/abs/2111.05992v2>] (ссылка на интернет-ресурс). Дата обращения: 25 января 2024 года.

21. Haarnoja, T., Tang, H., Abbeel, P., Levine, S. Reinforcement Learning with Deep Energy-Based Policies // Proceedings of the 34th International Conference on Machine Learning (ICML 2017). – 2017. – [<https://arxiv.org/abs/1702.08165>] (ссылка на интернет-ресурс). Дата обращения: 25 января 2024 года.

22. Unity Technologies. Training Configuration File. – [<https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-Configuration-File.md>] (ссылка на интернет-ресурс). Дата обращения: 20 декабря 2024 года.

23. Elo, A. The Rating of Chessplayers, Past and Present. – New York: Arco Publishing, 1978. – 206 p. – ISBN 0668-04711-6.

**ПРИЛОЖЕНИЕ А**  
**Листинг**  
(справочное)

## Листинг A.1 – Код файла MLGeneralUtilsEditor.cs

```
using UnityEngine;
using UnityEditor;
using System.IO;
using System.Collections.Generic;
using YamlDotNet.Serialization;
using YamlDotNet.Serialization.NamingConventions;

public class MLGeneralUtilsEditor : EditorWindow
{
    private string behaviorName = "MyBehavior";

    private enum TrainerType
    {
        PPO,
        SAC,
        Imitation
    }

    private TrainerType trainerType = TrainerType.PPO;

    // Hyperparameters
    private int batchSize = 1024;
    private int bufferSize = 10240;
    private float learningRate = 3e-4f;
    private float beta = 5e-4f;
    private float epsilon = 0.2f;
    private float lambda = 0.95f;
    private int numEpoch = 3;
    private string learningRateSchedule = "constant";

    // Network settings
    private bool normalize = false;
    private int hiddenUnits = 128;
    private int numLayers = 2;
    private string visEncodeType = "simple";
    private bool useMemory = false;
    private int memorySize = 128;
    private int sequenceLength = 64;

    // Reward signals
    private bool useCuriosity = false;
    private bool useGAIL = false;
    private bool useRND = false;

    private float gamma = 0.99f;
    private float extrinsicStrength = 1.0f;

    private float curiosityStrength = 0.02f;
    private float curiosityLearningRate = 3e-4f;
```

```

private float gailStrength = 0.5f;
private string gailDemoPath = "<path>";

private float rndStrength = 1.0f;
private float rndLearningRate = 3e-4f;

// Misc
private int keepCheckpoints = 5;
private int maxSteps = 50000;
private int timeHorizon = 64;
private int summaryFreq = 1000;
private bool threaded = true;

// Env Settings
private string envPath = null;
private string envArgs = "";
private int basePort = 5005;
private int numEnvs = 1;
private int envSeed = -1;
private bool noGraphics = false;

// Checkpoint Settings
private string runId = "my_run_id";
private string initializeFrom = null;
private bool loadModel = false;
private bool resume = false;
private bool force = false;
private bool trainModel = true;
private bool inference = false;

// Training History
private string trainingHistoryPath = "<history_path>";
private string savePath = "<savePath>";

[MenuItem("ML-Agents/Full Trainer Config Editor")]
public static void ShowWindow()
{
    GetWindow<MLGeneralUtilsEditor>("ML-Agents Full
Config");
}

private void OnGUI()
{
    GUILayout.Label("ML-Agents Trainer Config",
EditorStyles.boldLabel);

    behaviorName = EditorGUILayout.TextField("Behavior
Name", behaviorName);
    trainerType = (TrainerType)
EditorGUILayout.EnumPopup("Trainer Type", trainerType);

    GUILayout.Space(5);

```

```

        GUILayout.Label("Hyperparameters",
EditorStyles.boldLabel);
        batchSize = EditorGUILayout.IntField("Batch Size",
batchSize);
        bufferSize = EditorGUILayout.IntField("Buffer Size",
bufferSize);
        learningRate = EditorGUILayout.FloatField("Learning
Rate", learningRate);
        beta = EditorGUILayout.FloatField("Beta", beta);
        epsilon = EditorGUILayout.FloatField("Epsilon",
epsilon);
        lambd = EditorGUILayout.FloatField("Lambda", lambd);
        numEpoch = EditorGUILayout.IntField("Num Epoch",
numEpoch);
        learningRateSchedule = EditorGUILayout.TextField("LR
Schedule", learningRateSchedule);

        GUILayout.Space(5);
        GUILayout.Label("Network Settings",
EditorStyles.boldLabel);
        normalize = EditorGUILayout.Toggle("Normalize",
normalize);
        hiddenUnits = EditorGUILayout.IntField("Hidden Units",
hiddenUnits);
        numLayers = EditorGUILayout.IntField("Num Layers",
numLayers);
        visEncodeType = EditorGUILayout.TextField("Vis Encode
Type", visEncodeType);
        useMemory = EditorGUILayout.Toggle("Use Memory",
useMemory);
        if (useMemory)
        {
            memorySize = EditorGUILayout.IntField("Memory Size",
memorySize);
            sequenceLength = EditorGUILayout.IntField("Sequence
Length", sequenceLength);
        }

        GUILayout.Space(5);
        GUILayout.Label("Reward Signals",
EditorStyles.boldLabel);
        gamma = EditorGUILayout.FloatField("Gamma", gamma);
        extrinsicStrength =
EditorGUILayout.FloatField("Extrinsic Strength",
extrinsicStrength);

        useCuriosity = EditorGUILayout.Toggle("Use Curiosity",
useCuriosity);
        if (useCuriosity)
        {
            curiosityStrength =
EditorGUILayout.FloatField("Curiosity Strength",

```

```

curiosityStrength);
        curiosityLearningRate =
EditorGUILayout.FloatField("Curiosity LR",
curiosityLearningRate);
    }

    useGAIL = EditorGUILayout.Toggle("Use GAIL", useGAIL);
    if (useGAIL)
    {
        gailStrength = EditorGUILayout.FloatField("GAIL
Strength", gailStrength);
        gailDemoPath = EditorGUILayout.TextField("GAIL Demo
Path", gailDemoPath);
    }

    useRND = EditorGUILayout.Toggle("Use RND", useRND);
    if (useRND)
    {
        rndStrength = EditorGUILayout.FloatField("RND
Strength", rndStrength);
        rndLearningRate = EditorGUILayout.FloatField("RND
LR", rndLearningRate);
    }

    GUILayout.Space(5);
    GUILayout.Label("Other Settings",
EditorStyles.boldLabel);
    keepCheckpoints = EditorGUILayout.IntField("Keep
Checkpoints", keepCheckpoints);
    maxSteps = EditorGUILayout.IntField("Max Steps",
maxSteps);
    timeHorizon = EditorGUILayout.IntField("Time Horizon",
timeHorizon);
    summaryFreq = EditorGUILayout.IntField("Summary Freq",
summaryFreq);
    threaded = EditorGUILayout.Toggle("Threaded", threaded);

    GUILayout.Space(5);
    GUILayout.Label("Environment Settings",
EditorStyles.boldLabel);
    envPath = EditorGUILayout.TextField("Env Path",
envPath);
    envArgs = EditorGUILayout.TextField("Env Args (space-
separated)", envArgs);
    basePort = EditorGUILayout.IntField("Base Port",
basePort);
    numEnvs = EditorGUILayout.IntField("Num Envs", numEnvs);
    envSeed = EditorGUILayout.IntField("Seed", envSeed);
    noGraphics = EditorGUILayout.Toggle("No Graphics",
noGraphics);

    GUILayout.Space(5);

```

```

        GUILayout.Label("Checkpoint Settings",
EditorStyles.boldLabel);
        runId = EditorGUILayout.TextField("Run ID", runId);
        initializeFrom = EditorGUILayout.TextField("Initialize
From", initializeFrom);
        loadModel = EditorGUILayout.Toggle("Load Model",
loadModel);
        resume = EditorGUILayout.Toggle("Resume", resume);
        force = EditorGUILayout.Toggle("Force", force);
        trainModel = EditorGUILayout.Toggle("Train Model",
trainModel);
        inference = EditorGUILayout.Toggle("Inference",
inference);

        GUILayout.Space(5);
        GUILayout.Label("Training History",
EditorStyles.boldLabel);
        trainingHistoryPath =
EditorGUILayout.TextField("Training History Path",
trainingHistoryPath);

        GUILayout.Space(5);
        GUILayout.Label("Training History",
EditorStyles.boldLabel);
        savePath = EditorGUILayout.TextField("Save path",
savePath);

        GUILayout.Space(10);
        if (GUILayout.Button("Export YAML"))
        {
            ExportYaml();
        }

        if (GUILayout.Button("Learn"))
        {
            Learn();
        }
    }
    private void Learn()
    {

    }

    private void ExportYaml()
    {
        var config = new Dictionary<string, object>
        {
            ["trainer_type"] = trainerType.ToString().ToLower(),
            ["hyperparameters"] = new Dictionary<string, object>
            {

```

```

        ["batch_size"] = batchSize,
        ["buffer_size"] = bufferSize,
        ["learning_rate"] = learningRate,
        ["beta"] = beta,
        ["epsilon"] = epsilon,
        ["lambda"] = lambda,
        ["num_epoch"] = numEpoch,
        ["learning_rate_schedule"] =
learningRateSchedule,
    },
    ["network_settings"] = new Dictionary<string,
object>
    {
        ["normalize"] = normalize,
        ["hidden_units"] = hiddenUnits,
        ["num_layers"] = numLayers,
        ["vis_encode_type"] = visEncodeType,
        ["memory"] = useMemory
            ? new Dictionary<string, object>
            {
                ["memory_size"] = memorySize,
                ["sequence_length"] = sequenceLength
            }
            : null
    },
    ["reward_signals"] = BuildRewardSignals(),
    ["keep_checkpoints"] = keepCheckpoints,
    ["max_steps"] = maxSteps,
    ["time_horizon"] = timeHorizon,
    ["summary_freq"] = summaryFreq,
    ["threaded"] = threaded
};

var wrapped = new Dictionary<string, object>
{
    ["behaviors"] = new Dictionary<string, object>
    {
        [behaviorName] = config
    },
    ["env_settings"] = new Dictionary<string, object>
    {
        ["env_path"] = envPath,
        ["env_args"] =
            string.IsNullOrEmpty(envArgs) ? new
List<string>() : new List<string>(envArgs.Split(' ')),
        ["base_port"] = basePort,
        ["num_envs"] = numEnvs,
        ["seed"] = envSeed,
        ["no_graphics"] = noGraphics
    },
    ["checkpoint_settings"] = new Dictionary<string,
object>

```

```

        {
            ["run_id"] = runId,
            ["initialize_from"] =
string.IsNullOrEmpty(initializeFrom) ? null : initializeFrom,
            ["load_model"] = loadModel,
            ["resume"] = resume,
            ["force"] = force,
            ["train_model"] = trainModel,
            ["inference"] = inference
        }
    };

    var serializer = new SerializerBuilder()
        .WithNamingConvention(CamelCaseNamingConvention.Instance)
        .Build();

    string yaml = serializer.Serialize(wrapped);
    string path = EditorUtility.SaveFilePanel("Save Trainer
YAML", Application.dataPath, "trainer_config", "yaml");

    if (!string.IsNullOrEmpty(path))
    {
        File.WriteAllText(path, yaml);
        Debug.Log($"YAML saved to: {path}");
    }
}

private Dictionary<string, object> BuildRewardSignals()
{
    var rewardSignals = new Dictionary<string, object>
    {
        ["extrinsic"] = new Dictionary<string, object>
        {
            ["gamma"] = gamma,
            ["strength"] = extrinsicStrength
        }
    };

    if (useCuriosity)
    {
        rewardSignals["curiosity"] = new Dictionary<string,
object>
        {
            ["gamma"] = gamma,
            ["strength"] = curiosityStrength,
            ["learning_rate"] = curiosityLearningRate
        };
    }

    if (useGAIL)
    {

```

```

        rewardSignals["gail"] = new Dictionary<string,
object>
        {
            ["gamma"] = gamma,
            ["strength"] = gailStrength,
            ["demo_path"] = gailDemoPath
        };
    }

    if (useRND)
    {
        rewardSignals["rnd"] = new Dictionary<string,
object>
        {
            ["gamma"] = gamma,
            ["strength"] = rndStrength,
            ["learning_rate"] = rndLearningRate
        };
    }

    return rewardSignals;
}
}

```

## Листинг A.2 – Код файла LevelController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using AgentAI;
using Unity.MLAgents;
using UnityEngine;
using Random = UnityEngine.Random;

namespace Level
{
    public class LevelController : MonoBehaviour
    {
        [SerializeField]
        private FlagAgent[] _allTeam;

        [SerializeField]
        private FlagAgent[] _blueTeam;

        [SerializeField]
        private FlagAgent[] _redTeam;

        [SerializeField]
        private BaseController[] _baseControllers;

        [SerializeField]
        private FlagController _flagController;

        [SerializeField]
        private float _intervalStep;

        private SimpleMultiAgentGroup _blueTeamGroup;
        private SimpleMultiAgentGroup _redTeamGroup;

        private List<EnemyInfo> _enemyInfos = new
List<EnemyInfo>();
        private Vector3 _flagInitPosition;
        private float _timer;

        private void Cache()
        {
            _allTeam = GetComponentsInChildren<FlagAgent>();
            _blueTeam = _allTeam.Where(d =>
d.transform.CompareTag("blue")).ToArray();
            _redTeam = _allTeam.Where(d =>
d.transform.CompareTag("red")).ToArray();
            _flagController =
GetComponentInChildren<FlagController>();
            _baseControllers =
GetComponentsInChildren<BaseController>();
            foreach (FlagAgent flagAgent in _allTeam)
```

```

        {
            flagAgent.Configure(1f / _intervalStep);
            flagAgent.OnKilled += OnKilled;
        }
    }
    private void OnKilled(Team team)
    {
        if (_blueTeam.All(d => d.Dead))
        {
            WinRed(3);
            EndEpisodeAndStart();
            return;
        }

        if (_redTeam.All(d => d.Dead))
        {
            WinBlue(3);
            EndEpisodeAndStart();
            return;
        }
    }

    private void Awake()
    {
        Cache();
    }

    private void Start()
    {
        _blueTeamGroup = new SimpleMultiAgentGroup();
        _redTeamGroup = new SimpleMultiAgentGroup();
        _flagInitPosition =
        _flagController.transform.position;

        foreach (FlagAgent flagAgent in _blueTeam)
        {
            _blueTeamGroup.RegisterAgent(flagAgent);
            flagAgent.SetBase(_baseControllers.First(d =>
d.Team == Team.Blue), _flagController);
            _enemyInfos.Add(new EnemyInfo()
            {
                FlagAgent = flagAgent,
                InitPosition = flagAgent.transform.position,
                InitRotation = flagAgent.transform.rotation
            });
        }

        foreach (FlagAgent flagAgent in _redTeam)
        {
            _redTeamGroup.RegisterAgent(flagAgent);

```

```

        flagAgent.SetBase(_baseControllers.First(d =>
d.Team == Team.Red), _flagController);
        _enemyInfos.Add(new EnemyInfo()
        {
            FlagAgent = flagAgent,
            InitPosition = flagAgent.transform.position,
            InitRotation = flagAgent.transform.rotation
        });
    }

    Subscribe();
    ResetScene();
}

private void Subscribe()
{
    foreach (BaseController baseController in
_baseControllers)
    {
        baseController.OnFlagEnter += OnFlagEnter;
    }
}

private void OnFlagEnter(Team team)
{
    if (team == Team.Blue)
    {
        WinBlue(5);
    }
    else
    {
        WinRed(5);
    }

    EndEpisodeAndStart();
}

private void EndEpisodeAndStart()
{
    _blueTeamGroup.EndGroupEpisode();
    _redTeamGroup.EndGroupEpisode();
    ResetScene();
}

private void WinRed(int maxReward)
{
    _redTeamGroup.AddGroupReward(Mathf.Clamp(maxReward -
(float)_timer / _intervalStep, 1f, maxReward));
    _blueTeamGroup.AddGroupReward(-2);
}

private void WinBlue(int maxReward)
{

```

```

        _blueTeamGroup.AddGroupReward(Mathf.Clamp(maxReward
- (float)_timer / _intervalStep, 1f, maxReward));
        _redTeamGroup.AddGroupReward(-2);
    }

    private void FixedUpdate()
    {
        _timer += Time.fixedDeltaTime;
        if (_timer >= _intervalStep)
        {
            _blueTeamGroup.GroupEpisodeInterrupted();
            _redTeamGroup.GroupEpisodeInterrupted();
            ResetScene();
        }
    }

    private void ResetScene()
    {
        _timer = 0;
        _flagController.transform.position =
            _flagInitPosition + new
Vector3(Random.Range(-3f, 3f), 0, Random.Range(-3f, 3f));

        _flagController.transform.SetParent(transform);

        foreach (EnemyInfo enemyInfo in _enemyInfos)
        {
            Vector3 position = new Vector3(Random.Range(-3f,
3f), 0, Random.Range(-3f, 3f));

            enemyInfo.FlagAgent.transform.SetPositionAndRotation(enemyInfo.I
nitPosition + position,
                enemyInfo.InitRotation);

            enemyInfo.FlagAgent.ResetInfo();
        }

        _flagController.Taken = false;
    }

    private class EnemyInfo
    {
        public FlagAgent FlagAgent;
        public Vector3 InitPosition;
        public Quaternion InitRotation;
    }
}

```

### Листинг А.3 – Код файла FlagController.cs

```
using System;
using Level;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;

namespace AgentAI
{
    public class FlagAgent : Agent
    {
        [SerializeField]
        private float _runSpeed;

        [SerializeField]
        private float _slowDownSpeed;

        [SerializeField]
        private Rigidbody _rigidbody;

        [SerializeField]
        private Transform _rootTop;

        [SerializeField]
        private Team _team;

        [SerializeField]
        private RayPerceptionSensorComponent3D
_perceptionSensor;

        private EnvironmentParameters _resetParams;

        private BaseController _baseController;
        private FlagController _flagController;

        public Team Team => _team;
        public bool HasFlag { get; private set; }

        public bool Dead { get; private set; }
        public Rigidbody Rigidbody => _rigidbody;

        private float _forwardSpeed = 1.3f;
        private float _laterSpeed = 1.2f;
        private float _step;
        private float _flagCaptured;
        private float _currentSpeed;
        private float _holdingFlagReward;
        private float _deadPenalty;
        private float _killReward;
        private float _missAttackPenalty;
    }
}
```

```

private float _collisionPenalty;
private float _lastDistanceToFlag;
private float _rewardMoveToFlag;

public event Action<Team> OnKilled;

public void Configure(float step)
{
    _step = step;
}
public override void Initialize()
{
    base.Initialize();
    _resetParams =
Academy.Instance.EnvironmentParameters;
    HasFlag = false;
    _currentSpeed = _runSpeed;
}

private void FixedUpdate()
{
    if (Dead) return;
    if (_flagController == null) return;

    float distanceToFlag =
Vector3.Distance(transform.position,
_flagController.transform.position);

    // Награда за приближение к флагу
    if (!HasFlag)
    {
        float distanceDelta = _lastDistanceToFlag -
distanceToFlag;
        AddReward(distanceDelta * _rewardMoveToFlag);
        _lastDistanceToFlag = distanceToFlag;
    }
    else
    {
        float distanceToBase =
Vector3.Distance(transform.position,
_baseController.transform.position);
        float maxDistance = 150f;
        float proximityFactor = 1f -
Mathf.Clamp01(distanceToBase / maxDistance);
        float reward = _holdingFlagReward *
proximityFactor * Time.deltaTime;
        AddReward(reward);
    }
}

```

```

        public override void OnActionReceived(ActionBuffers
actions)
        {
            MoveAgent(actions.DiscreteActions);
            TryAttack(actions.DiscreteActions);
        }
        private void TryAttack(ActionSegment<int>
actionsDiscreteActions)
        {
            if (HasFlag) return;

            if (actionsDiscreteActions[3] == 0) return;

            float attackRange = 2.5f;
            Vector3 direction = transform.forward;
            RaycastHit hit;
            if (Physics.Raycast(transform.position, direction,
out hit, attackRange))
            {
                FlagAgent target =
hit.collider.GetComponent<FlagAgent>();
                if (target != null && target.Team != Team)
                {
                    target.Kill();
                    AddReward(_killReward);
                    return;
                }
            }

            AddReward(_missAttackPenalty);
        }
        private void Kill()
        {
            gameObject.SetActive(false);
            AddReward(_deadPenalty);
            Dead = true;
            OnKilled?.Invoke(_team);

            if (!HasFlag)
            {
                return;
            }

            _flagController.Taken = false;

            _flagController.transform.SetParent(_baseController.transform.pa
rent);

            _currentSpeed = _slowDownSpeed;
            HasFlag = false;
        }

        private void OnCollisionEnter(Collision collision)

```

```

        {
            var otherAgent =
collision.gameObject.GetComponent<FlagAgent>();
            if (otherAgent != null && otherAgent.Team == _team)
            {
                AddReward(_collisionPenalty);
            }
        }
    public void MoveAgent(ActionSegment<int> act)
    {
        var dirToGo = Vector3.zero;
        var rotateDir = Vector3.zero;

        //m_KickPower = 0f;

        var forwardAxis = act[0];
        var rightAxis = act[1];
        var rotateAxis = act[2];

        switch (forwardAxis)
        {
            case 1:
                dirToGo = transform.forward * _forwardSpeed;
                // m_KickPower = 1f;
                break;
            case 2:
                dirToGo = transform.forward *
- _forwardSpeed;
                break;
        }
        //TEST3 - worked
        switch (rightAxis)
        {
            case 1:
                dirToGo = transform.right * _laterSpeed;
                break;
            case 2:
                dirToGo = transform.right * -_laterSpeed;
                break;
        }

        switch (rotateAxis)
        {
            case 1:
                rotateDir = transform.up * -1f;
                break;
            case 2:
                rotateDir = transform.up * 1f;
                break;
        }

        transform.Rotate(rotateDir, Time.deltaTime * 100f);
    }

```

```

        _rigidbody.AddForce(dirToGo * _currentSpeed *
Time.deltaTime * 50,
        ForceMode.VelocityChange);
    }

    public override void Heuristic(in ActionBuffers
actionsOut)
    {
        var discreteActionsOut = actionsOut.DiscreteActions;
        //forward
        if (Input.GetKey(KeyCode.W))
        {
            discreteActionsOut[0] = 1;
        }

        if (Input.GetKey(KeyCode.S))
        {
            discreteActionsOut[0] = 2;
        }

        //rotate
        if (Input.GetKey(KeyCode.A))
        {
            discreteActionsOut[2] = 1;
        }

        if (Input.GetKey(KeyCode.D))
        {
            discreteActionsOut[2] = 2;
        }

        //right
        if (Input.GetKey(KeyCode.E))
        {
            discreteActionsOut[1] = 1;
        }

        if (Input.GetKey(KeyCode.Q))
        {
            discreteActionsOut[1] = 2;
        }

        if (Input.GetKey(KeyCode.Space))
        {
            discreteActionsOut[3] = 1;
        }
        else
        {
            discreteActionsOut[3] = 0;
        }
    }
}

```

```

        public void OnTriggerEnter(Collider other)
        {
            if (!
other.transform.TryGetComponent<FlagController>(out
FlagController flagController))
            {
                return;
            }

            if (flagController.Taken)
            {
                return;
            }

            flagController.Taken = true;
            flagController.transform.SetParent(_rootTop);
            flagController.transform.localPosition =
Vector3.zero;
            _flagController = flagController;
            AddReward(_flagCaptured);
            _currentSpeed = _slowDownSpeed;
            HasFlag = true;
        }

        public void ResetInfo()
        {
            Rigidbody.velocity = Vector3.zero;
            Rigidbody.angularVelocity = Vector3.zero;
            HasFlag = false;
            gameObject.SetActive(true);
            Dead = false;
            _currentSpeed = _runSpeed;
            _lastDistanceToFlag =
Vector3.Distance(transform.position,
_flagController.transform.position);
        }

        public override void OnEpisodeBegin()
        {
            _lastDistanceToFlag =
Vector3.Distance(transform.position,
_flagController.transform.position);
            _flagCaptured =
_resetParams.GetWithDefault("flag_captured", 0.1f);
            _rewardMoveToFlag =
_resetParams.GetWithDefault("reward_move_to_flag", 0.01f);
            _holdingFlagReward =
_resetParams.GetWithDefault("holding_flag_reward", 0.01f);
            _deadPenalty =
_resetParams.GetWithDefault("dead_penalty", -1);
            _killReward =

```

```

_resetParams.GetWithDefault("kill_reward", 0.5f);
    _missAttackPenalty =
_resetParams.GetWithDefault("miss_attack_penalty", -0.01f);
    _collisionPenalty =
_resetParams.GetWithDefault("collision_penalty", -0.001f);
    }

    public override void CollectObservations(VectorSensor
sensor)
    {
        base.CollectObservations(sensor);
        // Вектор до базы
        Vector3 dirToBase =
_baseController.transform.position - transform.position;
        sensor.AddObservation(dirToBase.normalized); // 3
флюата

        // Вектор до флага
        Vector3 dirToFlag =
_flagController.transform.position - transform.position;
        sensor.AddObservation(dirToFlag.normalized); // 3
флюата

        // Можно также добавить расстояния (как число)
        float distToBase =
Vector3.Distance(transform.position,
_baseController.transform.position);
        float distToFlag =
Vector3.Distance(transform.position,
_flagController.transform.position);
        sensor.AddObservation(distToBase); // 1 флюат
        sensor.AddObservation(distToFlag); // 1 флюат

        //sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(HasFlag ? 1f : 0f);
        //sensor.AddObservation(distanceToBase);
    }
    public void SetBase(BaseController baseController,
FlagController flagController)
    {
        _baseController = baseController;
        _flagController = flagController;
    }
}

```