Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий - РТФ
Кафедра информационных технологий и систем управления

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

Зав. кафедрой ИТиСУ
Е.В. Кислицын
(подпись) (Ф.И.О.)
«05» 06 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА СИСТЕМЫ ДЛЯ ОБУЧЕНИЯ ВИРТУАЛЬНЫХ СОЮЗНИКОВ НА БАЗЕ UNITY ML-AGENTS

Научный руководитель: Денисов Дмитрий Вадимович
к.т.н., доцент
подпись

Нормоконтролер: Бредихина Наталья Сергеевна
подпись

Студент группы: РИМ-220906 Копылов Данил Александрович
подпись

Екатеринбург
2024

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий - РТФ
Кафедра информационных технологий и систем управления
Направление подготовки 09.04.01 Информатика и вычислительная техника
Образовательная программа Инженерия искусственного интеллекта

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студента Копылова Данила Александровича группы РИМ-220906
(фамилия, имя, отчество)

1. Тема выпускной квалификационной работы Разработка системы для обучения виртуальных союзников на базе Unity ML-Agents
Утверждена распоряжением по институту от «4» декабря 2023 г. № 33.02-05/298

2. Научный руководитель Денисов Дмитрий Вадимович к.т.н, доцент
(Ф.И.О., должность, ученая степень, ученое звание)

3. Исходные данные к работе
предыдущие исследования и публикации по данной теме, описания методов и алгоритмов машинного обучения

4. Перечень демонстрационных материалов презентация MS PowerPoint

5. Календарный план

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	Обзор существующих методов обучения искусственного интеллекта в игровых симуляторах	до 23.03.2024 г.	Выполнено
2.	Изучение требований к разрабатываемой системе	до 29.04.2024 г.	Выполнено
3.	Реализация системы и проведение экспериментов для оценки эффективности	до 20.05.2024 г.	Выполнено
4.	ВКР в целом	до 24.05.2024 г.	Выполнено

Научный руководитель Денисов Дмитрий Вадимович (подпись)
Ф.И.О.

Студент задание принял к исполнению _____ (подпись)
Дата

6. Допустить Копылова Данила Александровича к защите выпускной квалификационной работы в экзаменационной комиссии

Зав. кафедрой ИТиСУ _____ Е.В. Кислицын
(подпись) Ф.И.О.

РЕФЕРАТ

Магистерская диссертация 69 с., 47 рис., 45 источн.

РАЗРАБОТКА СИСТЕМЫ ДЛЯ ОБУЧЕНИЯ ВИРТУАЛЬНЫХ СОЮЗНИКОВ НА БАЗЕ UNITY ML-AGENTS.

Ключевые слова: Игровой искусственный интеллект, машинное обучение, обучение с подкреплением, Unity ML-Agents, виртуальные агенты, Proximal Policy Optimization.

Объект исследования — методы и алгоритмы машинного обучения, применяемые для разработки и обучения виртуальных агентов.

Предметом исследования является процесс обучения виртуальных агентов в игровом окружении с использованием методов машинного обучения на платформе Unity ML-Agents.

Цель работы — разработка системы для обучения виртуальных союзников на базе Unity ML-Agents, способной демонстрировать различные уровни сложности поведения.

Методы исследования: теоретический анализ, анализ данных, алгоритмы машинного обучения, нейронные сети.

По итогу была разработана система для обучения виртуальных союзников на базе Unity ML-Agents. Были разработаны и протестированы четыре универсальных игровых сцены, каждая из которых демонстрирует определённый уровень сложности и тип поведения агентов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире видеоигры и игровые симуляторы становятся все более популярными и широко используемыми. Этот рост происходит параллельно с продолжающимся развитием технологий и доступностью высококачественных графических и виртуальных сред, которые создают уникальные и захватывающие игровые миры. Однако, помимо графики и игрового контента, игроки также ожидают улучшения в поведении неигровых персонажей (NPC), таких как виртуальные союзники. Включение машинного обучения в разработку видеоигр открывает новые возможности для создания более умных и реалистичных NPC, способных адаптироваться к игровой ситуации и реагировать на действия игрока с учетом контекста. Это открывает двери для создания игровых персонажей, которые демонстрируют более сложное поведение, эмоциональные реакции и стратегии. Сегодня искусственный интеллект играет ключевую роль в различных аспектах видеоигр, начиная от управления NPC и адаптации игрового уровня до создания реалистичных оппонентов в мультиплеерных сражениях. Тренды показывают, что игроки становятся все более требовательными к искусственному интеллекту в играх, желая более интеллектуальных и реалистичных взаимодействий со своими виртуальными союзниками и противниками. Качественные NPC играют важную роль в создании убедительного игрового мира и опыта игры. Виртуальные союзники, обладающие хорошо проработанным искусственным интеллектом, способны не только дополнять игровой процесс, но и создавать у игрока ощущение реального сотрудничества и взаимодействия. Благодаря играм можно не только просто развлекаться, но и выполнять рутинные задачи, а также обучаться чему-то новому. Так, с помощью различных симуляторов люди имеют возможность овладеть определенным навыком в безопасных условиях, но в то же время приближенных к реальным. Существуют разные

реабилитационные тренажеры, которые помогают быстрее восстановиться после травмы.

Доминирующим подходом в разработке игрового искусственного интеллекта является агентно-ориентированный подход, который включает в себя понятия агента и среды [1]. Агентом считается все, что может воспринимать среду и воздействовать на нее с помощью определенных исполнительных механизмов. Игровой агент взаимодействует со средой в соответствии с правилами игры, которые могут иметь разную сложность. Также свойства среды существенно влияют на поведение агента. При описании видеоигр с различными правилами и характеристиками игрового мира создаются модели миров произвольной сложности, для которых необходимо разработать эффективного и адаптивного агента. Таким образом, игры представляют отличное место для исследования и тестирования различных методов машинного обучения. Так, для обучения агентов часто используется обучения с подкреплением [2]. Обучение с подкреплением заключается в том, что агент взаимодействует с окружающей средой и получает от нее определенную обратную связь. В процессе обучения агент активно действует и интерпретирует обратную связь, которую получает в ответ на свои действия. Изначально агент не знает, какие шаги приведут к получению награды, но со временем, на основе полученного опыта, он выясняет, какие действия приносят наибольшее вознаграждение.

В обучении с подкреплением для того, чтобы задать архитектуру агента, используются нейронные сети. Так, после каждого выполненного шага агента, получив обратную связь, агент делает соответствующие выводы, корректируя веса нейронной сети для увеличения получаемой награды. Важно отметить, что агент не будет постоянно совершать определенное действие в зависимости от полученного отклика, но он его совершит с определенной вероятностью [3].

Актуальность данной темы обусловлена следующими факторами:

1) Рост ожиданий игроков: Современные игроки ожидают от игр высокой степени реализма и интерактивности. Они хотят видеть в играх

интеллектуальных персонажей, способных реагировать на их действия и адаптироваться к изменяющимся условиям.

2) Технологические достижения: Технологии машинного обучения, особенно обучение с подкреплением, предлагают новые методы для создания умных агентов, которые могут самостоятельно обучаться и улучшать свои навыки в процессе взаимодействия с игровой средой. Это открывает новые горизонты для разработчиков игр, позволяя им создавать более динамичные и сложные игровые миры [4].
3) Применение за пределами игровой индустрии: Методы и модели, разработанные для игровых ИИ, находят применение и в других областях, таких как робототехника, автоматизация и симуляции. Игровые агенты, способные к самообучению, могут быть использованы для моделирования поведения реальных систем, что делает эту область исследований особенно важной.
4) Конкурентное преимущество: В условиях высокой конкуренции на игровом рынке, инновационные решения, такие как использование ML-Agents, могут стать ключевым фактором успеха. Игры с более продвинутыми ИИ персонажами могут привлечь большее количество пользователей и повысить их удовлетворённость.

Целью данного исследования является разработка системы для обучения виртуальных союзников на базе Unity ML-Agents, способной демонстрировать различные уровни сложности поведения.

Объектом исследования в данной работе являются методы и алгоритмы машинного обучения, применяемые для разработки и обучения виртуальных агентов.

Предметом исследования является процесс обучения виртуальных агентов в игровом окружении с использованием методов машинного обучения на платформе Unity ML-Agents.

Для достижения цели необходимо решить следующие задачи:

1) Изучить существующие подходы к обучению виртуальных персонажей в играх.
2) Изучить возможности платформы Unity ML-Agents для реализации и обучения виртуальных агентов.
3) Разработать и реализовать систему для обучения виртуальных агентов.
4) Провести тестирование и анализ поведения агентов.

1 Анализ искусственного интеллекта в игровых симуляторах

1.1 Способы реализации игрового ИИ

В видеоиграх искусственный интеллект используется для выработки отзывчивого, адаптивного или интеллектуального поведения у неигровых персонажей. Искусственный интеллект был неотъемлемой частью видеоигр с момента их создания. Искусственный интеллект в видеоиграх – это отдельная область, которая отличается от академического искусственного интеллекта. Он служит для улучшения игрового процесса, а не для машинного обучения или принятия решений [5]. Во время золотого века аркадных видеоигр идея противников с искусственным интеллектом была в значительной степени популяризована в виде постепенных уровней сложности, четких схем движения и внутриигровых событий, зависящих от вклада игрока. Игровой ИИ в основном занимается выбором действий сущности в зависимости от текущих условий. Так, ИИ в игровой сфере отвечает за управление интеллектуальными агентами. Агентом обычно является персонаж игры, но это может быть и машина, робот или даже нечто более абстрактное – целая группа сущностей, страна или цивилизация. В любом случае это объект, следящий за своим окружением, принимающий на основании него решения и действующий в соответствии с этими решениями.

Игры уникальны тем, что для получения различной информации не требуется сложная система, так как эта информация уже встроена в симуляцию. Нет необходимости запускать алгоритмы распознавания изображений для обнаружения врага, поскольку игра уже знает о его присутствии и может непосредственно передать эту информацию в процесс принятия решений [6].

Основным принципом работы ИИ является принятие решений. Для того чтобы система могла принимать решения, она должна воздействовать на объекты с помощью искусственного интеллекта. Существует два способа

реализации. Первый заключается в том, что система ИИ изолирована в виде отдельного элемента игровой архитектуры. Такая стратегия зачастую принимает форму отдельного потока или нескольких потоков, в которых ИИ вычисляет наилучшее решение для заданных параметров. Второй способ реализуется благодаря тому, что объекты обращаются к искусственному интеллекту каждый раз, когда объект принимает решение. Такой подход отлично подходит для систем с большим количеством объектов, которым не требуется часто принимать решения.

Простейшей формой ИИ является система на основе правил. Данная система дальше всего стоит от настоящего искусственного интеллекта. Набор заранее заданных алгоритмов определяет поведение игровых объектов. С учетом разнообразия действий конечный результат может быть неявной поведенческой системой. Конечный автомат представляет собой метод моделирования и реализации объекта, который может находиться в различных состояниях на протяжении своей жизни. Каждое состояние может представлять физические условия, в которых находится объект.

В определенных случаях требуется, чтобы ИИ имел возможность развиваться, приспосабливаться и адаптироваться. Способность точно предугадывать следующий ход противника крайне важна для системы. Один из самых простых способов адаптации — отслеживание ранее принятых решений. ИИ регистрирует выбор, сделанный игроком в прошлом. Все принятые в прошлом решения необходимо оценить по каким-то критериям и характеристикам, а затем принять решение.

В последнее время стали популярны методы искусственного интеллекта и машинного обучения для реализации ИИ системы в видеоиграх для управления неигровыми персонажами и различной генерацией контента [7].

1.2 Обзор возможностей Unity ML-Agents

В работе описывается реализация искусственного интеллекта в компьютерных играх с использованием методов машинного обучения с подкреплением [8]. Unity ML-agents toolkit предоставляет разработчикам игр доступ к алгоритмам обучения с подкреплением, что позволяет создавать агентов с различными способностями. Было проведено сравнение эффективности обучения в разных средах, с различными типами вознаграждений. Результаты свидетельствуют о том, что сочетание внешних и внутренних вознаграждений ускоряет процесс обучения в среде с редкими вознаграждениями.

Обучающая среда — это место, где агент может наблюдать за окружающей средой и выполнять действия. Среда должна быть спроектирована таким образом, чтобы агент мог собрать достаточно информации, относящейся к его цели. Среда также должна содержать достаточное количество отрицательных или положительных вознаграждений для агента, чтобы он мог учиться, исследуя пространство действий. Агент должен уметь предсказывать, как его действия влияют на состояние среды, чтобы он мог учиться и понимать прогресс своего обучения. Агент влияет на свою окружающую среду с помощью своих действий, и его цель - найти наиболее прибыльное поведение, пробуя различные действия случайным образом. Для этого агент использует политику - функцию, которая определяет, какое действие следует предпринять в данном состоянии. В процессе обучения агент получает положительные или отрицательные вознаграждения как обратную связь за свои действия. Вознаграждения могут быть различными по частоте и размеру. Агент может получать отрицательные вознаграждения за ненадлежащее выполнение заданий и положительные вознаграждения за соответствующие действия. Во время обучения агент может получать как внешние, так и внутренние вознаграждения. Внутренние вознаграждения связаны с внутренней мотивацией агента, такой как любопытство, в то время

как внешние вознаграждения связаны с конкретными задачами в обучающей среде.

Градиентные методы основаны на оптимизации параметризованных политик с целью максимизации ожидаемого суммарного вознаграждения. Таким образом, целью является максимизация функции вознаграждения, поэтому используется алгоритм оптимизации градиентного спуска.

Алгоритм подкрепления Proximal Policy Optimization (PPO) является основным алгоритмом по умолчанию, используемым в инструментарии ML-agents. PPO - простой в использовании и относительно стабильный алгоритм, поскольку он защищен от разрушительных больших обновлений политики при градиентном спуске за счет ограничения максимального изменения обновления политики в меньшем диапазоне.

Вспомогательные методы обучения с подкреплением часто используются в качестве дополнения к алгоритму оптимизации проксимальной политики, поскольку они помогают агенту найти вознаграждение на ранних этапах обучения, что сокращает время обучения. Различные агенты обучаются с помощью следующих вспомогательных методов:

- оптимизация проксимальной политики;
- генеративно-состязательное имитационное обучение;
- дистилляция случайных сетей;
- исследование на основе любопытства;
- клонирование поведения.

Для тестирования методов подкрепления были созданы обучающие среды с плотным и редким вознаграждением. Среда с плотным вознаграждением представляет собой гоночную трассу, на которой агент получает вознаграждение за прохождение участков трассы и наказание за столкновение со стенами. Среда с редким вознаграждением — это город, в котором агент должен подобрать пассажиров и вернуть их в определенное

место. Агент получает вознаграждение только после выполнения всего задания. Важно разработать подходящую функцию вознаграждения, чтобы агент научился желаемому поведению. Если функция вознаграждения разработана неправильно, это может привести к нежелательным результатам или к тому, что агенты будут оптимизировать свое поведение для получения вознаграждения, а не для достижения намеченной цели. Было замечено, что агенты обучаются быстрее в среде с редким вознаграждением, если получают комбинацию внешних и внутренних вознаграждений. Рекомендуется использовать RND для редких вознаграждений, а методы имитационного обучения для имитации поведения игрока.

В работе проводится проектирование, разработка и тестирование системы, использующей алгоритм Proximal Policy Optimization для обучения искусственных агентов в многопользовательском шутере от первого лица [9]. Основная задача заключается в формировании команд, состоящих полностью из искусственных игроков, способных эффективно действовать в игровом мире. Основное внимание уделено оценке эффективности работы агентов в сложных ситуациях и выявлению областей, требующих дальнейшего улучшения, особенно в условиях конкуренции и сотрудничества. Исследование также включает изучение применения глубокого обучения с подкреплением и рекуррентных нейронных сетей для обеспечения агентов "памятью" и повышения качества принятия решений в реальном времени [10].

В статье рассматривается применение глубокого обучения с подкреплением и виртуальной реальности для обучения виртуальной руки робота, которая будет направлять пользователей во время выполнения физических упражнений [11]. Исследование предлагает новую игровую механику, основанную на искусственном интеллекте, использующую глубокое обучение с подкреплением для визуальной поддержки пользователей в движениях во время физических упражнений в иммерсивной виртуальной реальности. Представленные результаты демонстрируют эффективность обученных агентов и их способность предоставлять пользователям

уникальную информацию. Представленные результаты демонстрируют высокую скорость обучения благодаря параллельному обучению 16 агентов в течение миллиона шагов. Обученные модели способны соревноваться с человеческими навыками в виртуальной реальности и выполнять некоторые упражнения уже после одной тренировки.

Представленная система основана на модифицированной версии игры "Project Butterfly" (PBF) и предназначена для физических упражнений для верхних конечностей под управлением искусственного интеллекта. В системе использован игровой движок Unity и VR-система HTC Vive Pro, которая работает на основе внешнего отслеживания с помощью лазерных систем. Это позволяет собирать данные о позах в 3D-пространстве [12]. HTC Vive уже была использована в исследованиях для терапевтической геймификации и анализа постурального анализа. Главная цель игры - защищать виртуальную бабочку от неблагоприятных погодных условий и атаки, используя виртуальный "пузырьковый щит", который игрок активирует с помощью HTC Vive.

Проект IB был интегрирован с Unity ML-Agents, позволяющим обучать интеллектуальных агентов в игровой среде [13]. Проект использует крутящий момент и угловой момент в верхних конечностях для прогнозирования движений и силы, создаваемой мышцами. Предыдущие исследования показали, что эти показатели могут быть полезны как для реабилитации различных атлетических групп, так и для оценки эффективности движений. Модель искусственного интеллекта используется для прогнозирования среднего крутящего момента и углового момента. Агенты обучаются выполнять задачу с помощью обратной связи и наблюдения за динамикой суставов. Три упражнения были выбраны для обучения агентов: горизонтальное вращение плеча, поднятие руки вперед и поднятие руки вбок. Для обучения использовались два алгоритма: Proximal Policy Optimization (PPO) и Generative Adversarial Imitation Learning (GAIL). PPO позволяет агенту выбирать оптимальное действие на основе взаимодействия с окружающей

средой, а GAIL обеспечивает возможность имитировать упражнения пользователя и усиливает сигнал вознаграждения агента.

В статье представлено исследование, посвященное созданию и обучению виртуального двойника робота-манипулятора с использованием искусственного интеллекта в виртуальной среде, а также применению имитационного обучения в реальном мире [14]. В работе показаны рекомендации по протоколам обучения для виртуального двойника и подробности о необходимой архитектуре для эффективного моделирования в виртуальном пространстве. Также подчеркивается важность корректировки гиперпараметров и разработки эффективной системы вознаграждения для успешного обучения. Показана архитектура нейронной сети для обучения, включая плотные слои с функциями активации и масками действий. Процесс обучения включал создание виртуальной среды с помощью Unity и ее связь с физическим миром с использованием 3D-печатной копии руки робота. Сочетание использования игрового движка Unity и инструмента машинного обучения Google TensorFlow позволяет реализовать имитационное обучение, поведенческое клонирование, генеративно-состязательное имитационное обучение и обучение по учебному плану [15]. Все эти инструменты и опции позволяют расширить сферу применения для настройки политики обучения для различных задач.

Для экспериментальной установки вокруг руки робота были установлены шесть платформ, которые расположены на равном расстоянии от захвата руки. Кубики белого цвета используются для размещения на них других кубиков. Красные и зеленые кубы являются основной целью для робота, которую он должен найти, захватить, переместить и отпустить на другую незанятую платформу. Задача, которую предстоит решить симулятору, заключается в том, чтобы суметь взять цветной кубик и поместить его на другую незанятую опору. Обучение робота-манипулятора производилось с использованием алгоритма обратного распространения ошибки. Для этого роботу понадобилось более 30 часов. Однако с течением

времени эффективность робота будет повышаться и скомпенсирует начальные временные затраты. Увеличение вычислительной мощности значительно ускорить процесс обучения.

В статье подробно изучается развитие глубокого обучения с подкреплением в игровой индустрии, особенно в жанре шутеров от третьего лица [16]. Основное внимание уделено обучению агентов, которые применяют алгоритмы оптимизации проксимальной политики в среде Unity ML-Agents [17]. Агенты обучаются различным навыкам, таким как поиск врагов, сбор патронов, пополнение боеприпасов и атака врагов. Исследования показывают, что глубокое обучение существенно улучшает уровень интеллекта агентов по сравнению с традиционным интеллектуальным поведением. Также рассматривается структурная схема ML-агентов, состоящая из четырех частей: сбор наблюдений, действия агента, сброс агента и настройка системы вознаграждений.

Работа рассматривает применение видеоигр в качестве обучающей платформы для разработки и экспериментов с алгоритмами машинного обучения, с фокусом на наборе инструментов Unity ML-Agents [18]. Основной целью исследования было создание интегрируемых и высокопроизводительных агентов, способных решать задачи передвижения. Авторы исследования подробно описывают функциональные возможности и ключевые аспекты создания надежных моделей поведения для интеграции агентов машинного обучения в видеоигры. Исследование оценивает эффективность различных моделей нейронных сетей, обученных с использованием метода оптимизации проксимальной политики и различных конфигураций обучения. Также в статье обсуждается система вознаграждения, разработанная для минимизации времени на обучение и поощрения агентов за успешное достижение целей. Однако новые тестовые данные показали, что наиболее эффективные конфигурации не достигли требуемого уровня успеха, что указывает на необходимость более гибких моделей поведения для успешного решения задач в будущем [19].

В качестве метода обучения SAC использует буфер воспроизведения, который позволяет агентам вспоминать прошлый опыт во время обучения. Он также претендует на звание алгоритма максимального вознаграждения и энтропии. То есть агенты будут успешно справляться с поставленными задачами, действуя при этом как можно более случайным образом. Было решено ограничиться только одним из методов обучения с подкреплением, предлагаемых инструментарием. В документации к инструментарию было отмечено, что PPO является более универсальным и стабильным алгоритмом, и многие проекты стали использовать его как наиболее предпочтительный метод. Также было отмечено, что SAC имеет некоторые проблемы с производительностью на некоторых системах

Обучение по учебному плану — это метод обучения агентов путем разбиения больших задач на более мелкие и управляемые и постепенного повышения сложности до желаемого уровня. Этот подход позволяет агентам поэтапно осваивать сложное поведение и способствует плавному обучению. Реализация этого метода требует возможности изменять переменные окружения, которые формулируют задачу во время выполнения. В данном случае это означает сложное манипулирование данными во время обучения.

Проводится исследование влияния различных наблюдаемых характеристик на производительность агента в игре с мотоциклом. Отмечается, что позиция и расстояние до цели не оказывают существенного влияния на результаты игры, так как агент мотивирован двигаться вперед независимо от своего положения и расстояния до цели. Однако добавление информации о возвышении агента, угле агента относительно горизонтальной плоскости и наклоне трассы под агентом приводит к улучшению и стабильности работы агента. Предоставление информации о местности под каждым колесом также повышает принятие решений. Было также исследовано использование датчиков формирования лучей, но результаты всех тестов были схожи. Кроме того, применение лучевой передачи иногда блокировалось кривизной местности, поэтому был разработан собственный датчик,

способный считывать точки вдоль трассы независимо от ее структуры. Это решение улучшило результаты обучения. Изначально агенты получали награды за достижение определенных контрольных точек во время обучения. Каждая контрольная точка давала больше очков, чем предыдущая, и идея заключалась в том, чтобы неявно научить агентов двигаться вперед, чтобы достичь цели. Такое редкое вознаграждение приводило к медленному обучению, и агенты часто застревали, выполняя случайные действия бесконечно. Вместо этого агенты стали получать небольшое вознаграждение за каждый шаг, если они каким-то образом продвигались вперед. Это привело к заметному продвижению агента по уровню и улучшило процесс обучения

Процедурная генерация контента — мощный инструмент в игровой индустрии, но он может привести к отсутствию контроля и неправильной характеристике игрового дизайна. В статье предлагается стратегия процедурного создания подземелий с использованием методов машинного обучения [20]. В сгенерированных подземельях учитываются особенности расположения комнат и сохранены игровые характеристики, благодаря чему получаются более надежные карты, которые приятнее и удобнее воспроизводить по сравнению с картами, созданными вручную.

Метод генерации подземелий, предложенный в данной работе, использует агентов для прокладки путей к комнатам и принятия решений о размещении стратегических комнат. Методы машинного обучения, доступные в библиотеке Unity ML-Agents, используются для повышения интеллекта генерирующего агента [21]. Внедрение включает методы наблюдения за восприятием окружающей среды, методы действий агентов и методы поощрения за предоставление вознаграждений или наказаний. Полученные подземелья имеют формы, напоминающие пещеры или лесные лабиринты, и включают комнаты, расположенные случайным образом. Начальная комната подземелья выбирается случайным образом в матрице, чтобы создать больше путей и увеличить площадь подземелий. Разные типы комнат, такие как комнаты с боссами, комнаты для приобретения снаряжения, магазины и

комнаты с сокровищами, расположены в соответствии с определенными правилами проектирования.

В работе обсуждается внедрение обучения с подкреплением с использованием Unity3D и инструментария ML-Agents [22]. Авторы используют нейронные сети и алгоритмы временных разностей для обучения сети в настольной игре, представляющей собой более сложную версию игры «Крестики-нолики». TensorFlow и TensorFlowSharp используются для внешней тренировки мозга агента и комбинируют модель TensorFlow с Unity3D [23]. Использование Unity3D и инструментария ML-Agents позволяет внедрить обучение с подкреплением в разработку игр. Процесс обучения включает создание среды, получение положительных и отрицательных вознаграждений за действия агента и постоянное повышение эффективности агента с помощью итераций.

В игре «Крестики-нолики» обучение с подкреплением реализуется с использованием алгоритма SARSA, который позволяет системе быстро менять стратегии и учиться, взаимодействуя с противниками. Было обнаружено, что этому алгоритму требуется меньше времени для обучения и победы по сравнению с Q-обучением.

2 Обзор алгоритмов обучения

2.1 Unity ML-Agents

Unity Machine Learning Agents Toolkit — это проект с открытым исходным кодом, который позволяет играм и симуляциям выступать в качестве обучающих сред для интеллектуальных агентов. Инструментарий предоставляет удобный Python API для обучения агентов с использованием методов обучения с подкреплением, имитационного обучения, нейроэволюции и других методов машинного обучения [24]. Эти обученные агенты могут применяться для различных целей, включая управление NPC в различных настройках, таких как многопользовательские и состязательные игры, автоматизацию тестирования игровых сборок и оценку различных дизайнерских решений перед выпуском. Инструментарий ML-Agents предоставляет ценную платформу как для разработчиков игр, так и для исследователей искусственного интеллекта. Он позволяет оценивать достижения в области искусственного интеллекта в среде Unity, делая их доступными для более широкого сообщества исследователей и разработчиков игр [25]. На рисунке 1 изображены компоненты ML-Agents.

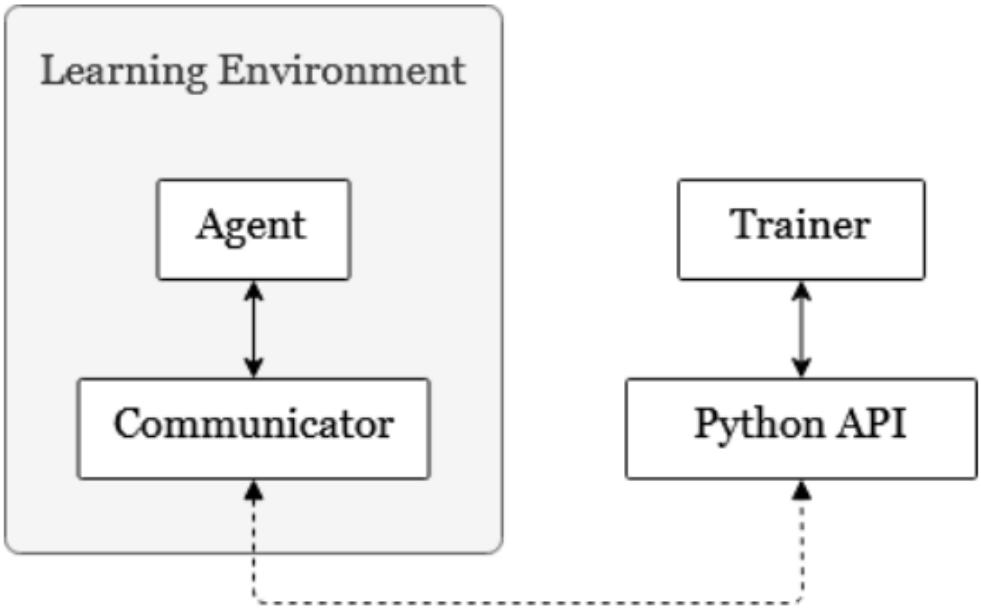


Рисунок 1 - Базовые компоненты ML-Agents

Есть 4 основных компонента, которые связаны между собой. Учебная среда включает в себя сцену и все игровые объекты, связанные с процессом обучения. Python API - отвечает за взаимодействие и манипулирование учебной средой через низкоуровневый интерфейс. Communicator соединяет низкоуровневый Python API с учебной средой. Trainer содержит алгоритмы машинного обучения, которые позволяют обучать агентов, взаимодействуя только через низкоуровневый Python API.

Наиболее распространённым подходом для обучения агентов является обучение с подкреплением (Reinforcement Learning). Это метод машинного обучения, при котором агент обучается методом проб и ошибок [26]. Агент взаимодействует со средой, обучаясь параллельно, и получает вознаграждение за выполнение действий. В обучении с подкреплением используется система наград: положительные награды за правильные действия и отрицательные за неправильные. Это побуждает агента выполнять желаемые действия и избегать нежелательных. Метод программирует агента на поиск долгосрочного и максимального общего вознаграждения, стремясь к оптимальному решению. Эти долгосрочные цели не позволяют агенту

останавливаться на достигнутом. Со временем система учится избегать негативных действий и совершать только позитивные. Обучение происходит на основе взаимодействия с окружающей средой методом проб и ошибок.

Область обучения с подкреплением включает несколько алгоритмов, использующих различные подходы. Эти алгоритмы различаются в основном стратегиями взаимодействия с окружающей средой.

State-Action-Reward-State-Action (SARSA) – это алгоритм обучения с подкреплением, который начинается с предоставления агенту определённой политики. Политика представляет собой вероятность, с помощью которой алгоритм оценивает шансы того, что определённые действия приведут к вознаграждениям или положительным состояниям [27].

В методе Q-Learning применяется противоположный подход. Здесь агент не получает политики, и его исследование окружающей среды осуществляется более независимо. В Q-Learning отсутствуют ограничения на выбор действий для алгоритма. Он предполагает, что все последующие выборы действий будут оптимальными по умолчанию, поэтому алгоритм осуществляет выбор, ориентируясь на максимизацию оценки Q.

Deep Q-Networks — это алгоритм, который использует нейронные сети в сочетании с методами обучения с подкреплением. Нейронные сети самостоятельно исследуют среду обучения с подкреплением для выбора оптимальных значений. Алгоритм основывается на выборке прошлых положительных действий, полученных от нейронной сети, и на их основе определяет, как себя вести и какие значения подбирать.

Так же популярен метод обучения, который называется нейроэволюция. Это форма искусственного интеллекта, которая использует эволюционные алгоритмы для генерации искусственных нейронных сетей, параметров и правил [28]. Чаще всего применяется в искусственной жизни, обычных играх и эволюционной робототехнике. Главное преимущество заключается в том, что нейроэволюция может применяться более широко, чем алгоритмы контролируемого обучения, для которых требуется программа из правильных

пар ввода-вывода. Напротив, нейроэволюция требует только оценки производительности сети при выполнении задачи. Нейроэволюция обычно используется как часть парадигмы обучения с подкреплением, и ее можно противопоставить традиционным методам глубокого обучения, которые используют обратное распространение (градиентный спуск в нейронной сети) с фиксированной топологией

2.2 SARSA

SARSA — это алгоритм обучения с подкреплением, который используется для решения задачи управления марковским процессом принятия решений [29]. Он основан на идее обучения с подкреплением, где агент взаимодействует с окружающей средой, выбирая действия и получая награды или штрафы за свои действия. Цель SARSA заключается в обучении агента выбирать оптимальные действия в различных состояниях среды с целью максимизации общей полученной награды.

В ходе каждого шага обучения алгоритма происходит последовательность действий: выбор действия, переход в новое состояние, получение награды, выбор нового действия и обновление оценки значения действия [30]. Исходя из этой последовательности, агент оценивает значение для каждой пары состояние-действие и обновляет их на каждом шаге. Основными принципами алгоритма являются марковский процесс принятия решений и Q-функция с ее обновлением.

Марковский процесс принятия решений (MDP) – это математическая модель, которая используется для формализации задачи управления в условиях неопределенности. MDP включает в себя состояния, действия, функцию перехода, функцию награды и временной горизонт. SARSA основан на концепции MDP, где агент принимает решения на основе текущего состояния и выбирает действия для максимизации общей полученной награды в течение определенного временного периода. Q-функция в MDP представляет

собой оценку значения состояния-действия. Она определяет ожидаемую общую получаемую награду в будущем при выборе определенного действия в текущем состоянии. В SARSA-алгоритме Q-функция обновляется на основе выбранного действия, следующего состояния, полученной награды и нового выбранного действия.

Главные этапы алгоритма заключаются в том, что сначала происходит инициализация Q-функции, где начальные значения устанавливаются случайными или нулевыми [31]. Затем агент наблюдает за состоянием окружающей среды, выбирает действие на основе стратегии выбора и выполняет взаимодействие со средой. Далее агент отслеживает новое состояние и получает соответствующую награду, после этого происходит обновление функции.

Перед тем, как приступить к реализации SARSA, необходимо определить функцию награды, которая будет использоваться для оценки действий агента. Эта функция должна быть выбрана таким образом, чтобы помогать агенту достигать его конечной цели и стимулировать желательные действия. Также необходимо определить модель состояний, которая будет отражать окружение агента и предоставлять информацию о его текущем состоянии.

Одним из ключевых аспектов реализации является выбор стратегии обновления Q-функции. Q-функция является оценочной функцией, которая отображает пару состояние-действие на ожидаемую общую награду от выполнения этого действия в данном состоянии. В SARSA обновление Q-функции происходит на основе текущего состояния, выбранного действия, полученной награды, нового состояния и выбранного в нем нового действия. Существуют различные подходы к обновлению Q-функции, такие как метод временной разности или градиентный спуск.

Одной из сложностей обучения с подкреплением является задача балансировки между исследованием и эксплуатацией. Агент должен исследовать окружающую среду, чтобы определить оптимальные действия, но

при этом он также должен использовать полученный опыт для максимизации общей награды. Для этого часто используется эпсилон-жадная стратегия, которая позволяет агенту с некоторой вероятностью выбирать случайное действие (исследование) и с вероятностью 1-epsilon - выбирать действие с максимальной ожидаемой наградой (эксплуатация). Такой подход позволяет достигать баланса между исследованием и использованием накопленного знания [32].

SARSA является одним из наиболее эффективных алгоритмов в области обучения с подкреплением. Этот алгоритм позволяет агенту учиться на основе взаимодействия с окружающей средой, принимая решения и обновляя значения Q-функции на каждом шаге. Такой подход позволяет агенту постепенно улучшать свою стратегию, максимизируя получаемую награду. SARSA широко применяется в различных игровых сценариях, где агент должен принимать решения на основе текущего состояния игры. Этот алгоритм также используется для обучения искусственных соперников, позволяя им принимать оптимальные решения и адаптироваться к условиям игры. В области робототехники SARSA применяется для обучения агентов-роботов принимать решения на основе текущего состояния окружающей среды. Этот алгоритм помогает роботам учиться на основе взаимодействия с окружением и находить оптимальные действия для достижения поставленных целей, таких как навигация в неизвестной среде или выполнение манипуляционных задач.

Преимущества SARSA:

- универсальность;
- итеративное обучение;
- способность к обновлению;
- работа с неполной информацией.

2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) — это алгоритм обучения с подкреплением, разработанный для улучшения стабильности и эффективности процесса обучения политики. PPO является одним из семейства алгоритмов оптимизации политики, который использует некоторые идеи из предшествующего алгоритма TRPO (Trust Region Policy Optimization) для достижения баланса между простотой реализации и сильной производительностью в разнообразных задачах [33]. PPO работает, оптимизируя целевую функцию, которая максимизирует прибыль, одновременно ограничивая степень изменения политики на каждом шаге обновления. Это обеспечивает стабильность обучения и предотвращает проблемы, возникающие при больших обновлениях политики [34]. Рассмотрим ключевые шаги PPO, показанные на рисунке 2. Используя текущую политику, агент взаимодействует с окружением и собирает данные о состояниях, действиях и вознаграждениях. Оценка функции преимущества для каждого действия, основанная на разнице между полученным вознаграждением и ожидаемым вознаграждением согласно оценочной функции состояния. Затем вычисляется суррогатная функция прибыли, которая измеряет ожидаемую эффективность действий по сравнению с базовой политикой. После оптимизируется эта функция с использованием методов градиентного подъема для обновления параметров политики. Чтобы избежать слишком больших изменений в политике, PPO вводит механизм клиппинга. Если отношение новой и старой политики выходит за пределы заданного интервала, прибыль обрезается, что ограничивает эффект обновления. Процесс повторяется с обновленной политикой, собирая новые данные и повторяя оптимизацию.

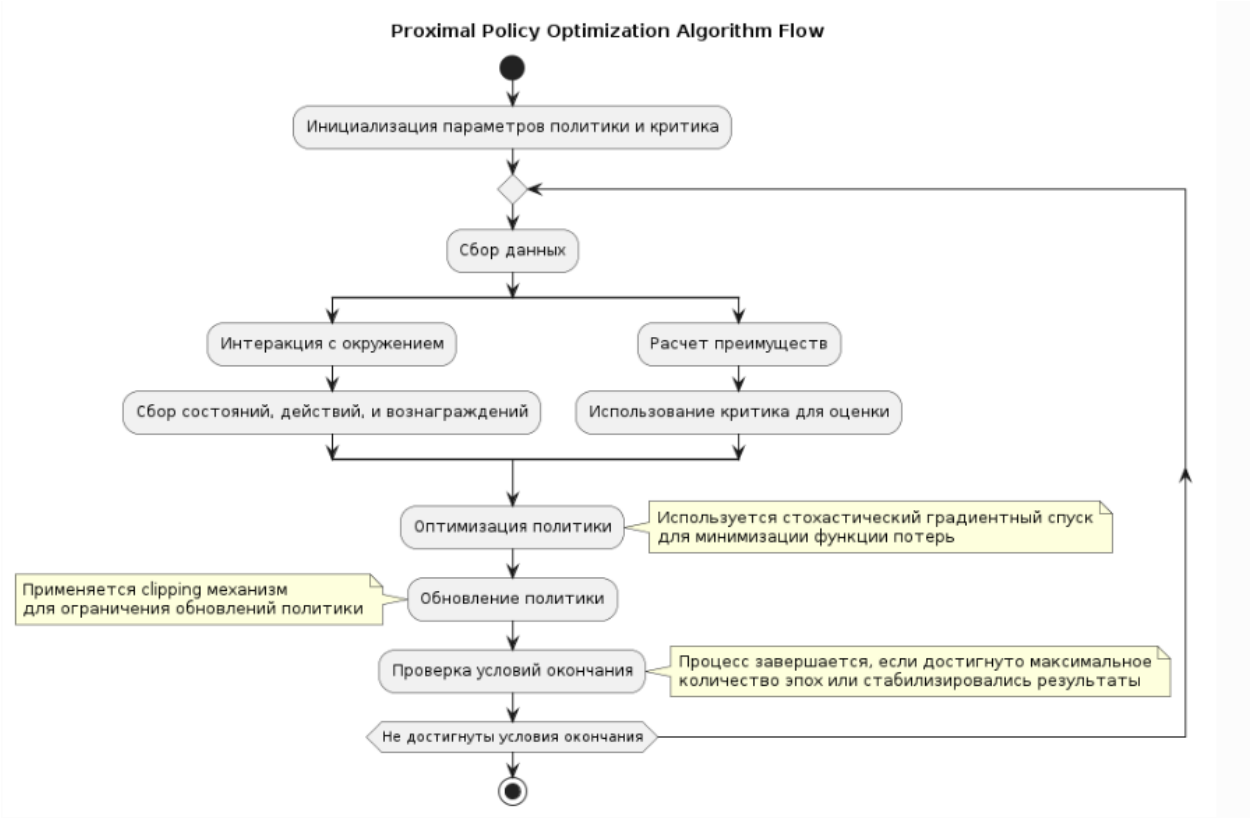


Рисунок 2 - Ключевые шаги алгоритма

В алгоритме ключевым понятием является политика или стратегия, которая определяет, какие действия агент должен предпринимать в ответ на состояния среды. Обычно политика представлена нейронной сетью. PPO оптимизирует функцию потерь, которая состоит из нескольких компонентов. Основными из них являются функция потерь для политики (policy loss) и функция потерь для оценки значений состояний (value loss). PPO стремится улучшить текущую политику, делая это в "проксимальном" смысле, чтобы гарантировать стабильность обучения. Для этого ограничиваются изменения в политике [35].

2.4 Soft Actor-Critic

Основной целью SAC является максимизация ожидаемой награды с учетом максимальной энтропии политики, что позволяет находить

разнообразные оптимальные решения в стохастических средах [36]. Soft Actor-Critic использует Актера со стохастической политикой. Это означает, что Актер в состоянии может выбрать некое действие из всего пространства действий с какой-то вероятностью. Таким образом, политика Актера в каждом конкретном состоянии позволяет выбрать не одно конкретное оптимальное действие, а любое из возможных действий, но с определенной долей вероятности. И в процессе обучения Актер учит это вероятностное распределения получения максимального вознаграждения. Это свойство стохастической политики Актера позволяет исследовать различные стратегии и обнаруживать оптимальные решения, которые могут быть скрыты при использовании детерминированной политики. Кроме того, стохастическая политика Актера учитывает неопределенность в окружающей среде. При наличии шума или случайных факторов такая политика может быть более устойчивой и адаптивной, поскольку позволяет генерировать разнообразные действия для эффективного взаимодействия с окружающей средой. Однако, обучение стохастической политики актера вносит коррективы и в процесс обучения [37]. Классическое обучение с подкреплением направлено на максимизацию ожидаемой доходности. В процессе обучения, можно сказать, для каждого действия мы выбираем такое действие, которое с наибольшей вероятностью даст нам большую доходность. SAC не использует целевой модели Актера. Для выбора действия в текущем и последующем состоянии используется одна обучаемая модель Актера. Тем самым подчеркивается, что достижение будущего вознаграждения достигается с использование текущей политики [38]. Кроме того, использование одной модели Актера позволяет снизить затраты памяти и вычислительных ресурсов.

2.5 Искусственные нейронные сети

Искусственная нейронная сеть (ИНС) представляет собой математическую модель или ее программное и аппаратное воплощение,

построенное по принципу организации биологических нейронных сетей. ИНС состоит из соединенных и взаимодействующих между собой простых процессоров, называемых искусственными нейронами. Каждый нейрон в такой сети работает с сигналами, которые он получает и передает другим нейронам. Хотя отдельные нейроны могут быть простыми, их совокупное взаимодействие в большой сети позволяет выполнять сложные задачи при правильной настройке и управлении. (рисунок 3).

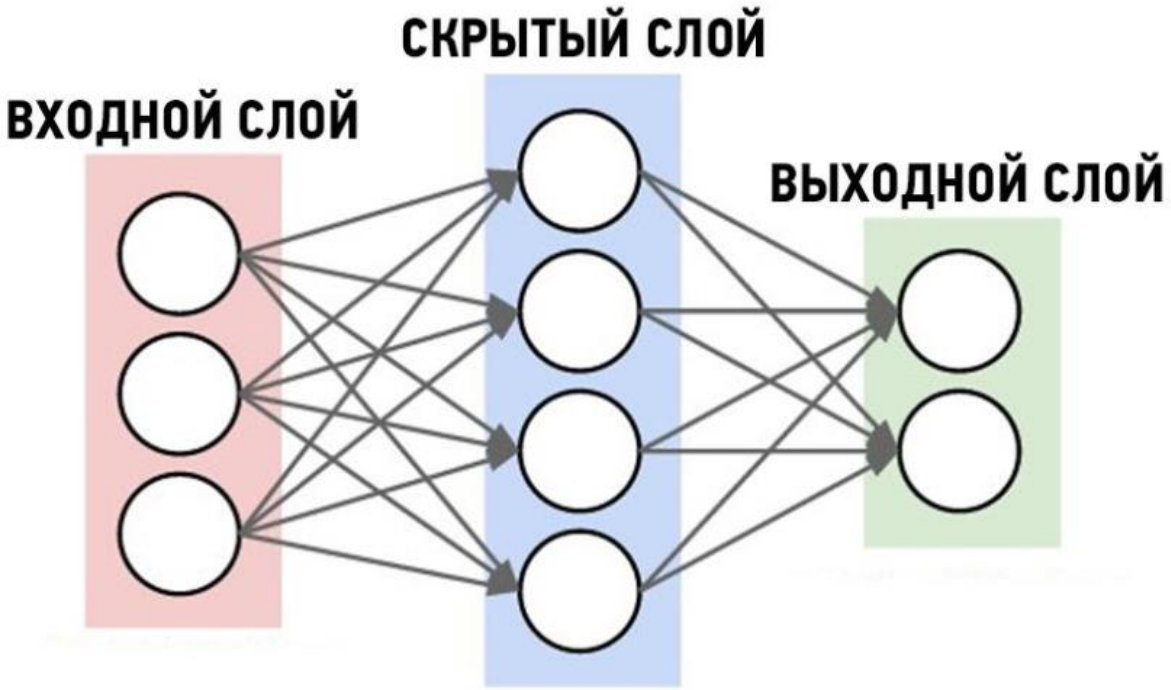


Рисунок 3 - Простая нейронная сеть

Обучение с учителем считается наиболее простым благодаря отсутствию необходимости в разработке алгоритмов самообучения. Для этого типа обучения требуется размеченный и структурированный набор данных. Под "размеченным" понимается процесс присвоения правильных ответов конкретным входным данным, ожидаемым от нейронной сети в качестве результата. За счет такой разметки нейронная сеть ориентируется на правильные ответы и корректирует свой процесс обучения, опираясь на

совершенные ошибки, выявленные сравнением размеченных данных с ее собственными прогнозами.

Обучение без учителя представляет собой процесс обучения нейронной сети без вмешательства или контроля со стороны человека. В этом случае нейронная сеть самостоятельно находит корреляции в данных, извлекает полезные признаки и проводит их анализ. Существуют несколько методов обучения без учителя: обнаружение аномалий, ассоциации, использование автоэнкодеров и кластеризация. В процессе обучения на основе этих методов нейронная сеть оценивает ошибку на каждом этапе и корректирует процесс обучения. Этот подход делает обучение без учителя самым сложным, поскольку написание алгоритмов для таких моделей требует много времени и усложняет процесс моделирования нейронной сети.

Обучение с подкреплением или частичным вмешательством учителя является своеобразной золотой серединой в обучении нейронных сетей [39]. Этот подход сочетает в себе основные принципы обучения без учителя с периодическим вмешательством учителя для коррекции процесса. Корректировка обучения происходит тогда, когда нейронная сеть допускает ошибки в процессе обучения.

3 Этапы обучения и поведения виртуальных агентов

3.1 Простое поведение агента для достижения цели

Для изучения возможностей Unity ML-Agents была реализована простая сцена в Unity 3d (рисунок 4). Она включает в себя область, в которой находятся следующие дочерние объекты: агент, награда и пол. Цель сцены заключается в том, чтобы агент обучился собирать награду.

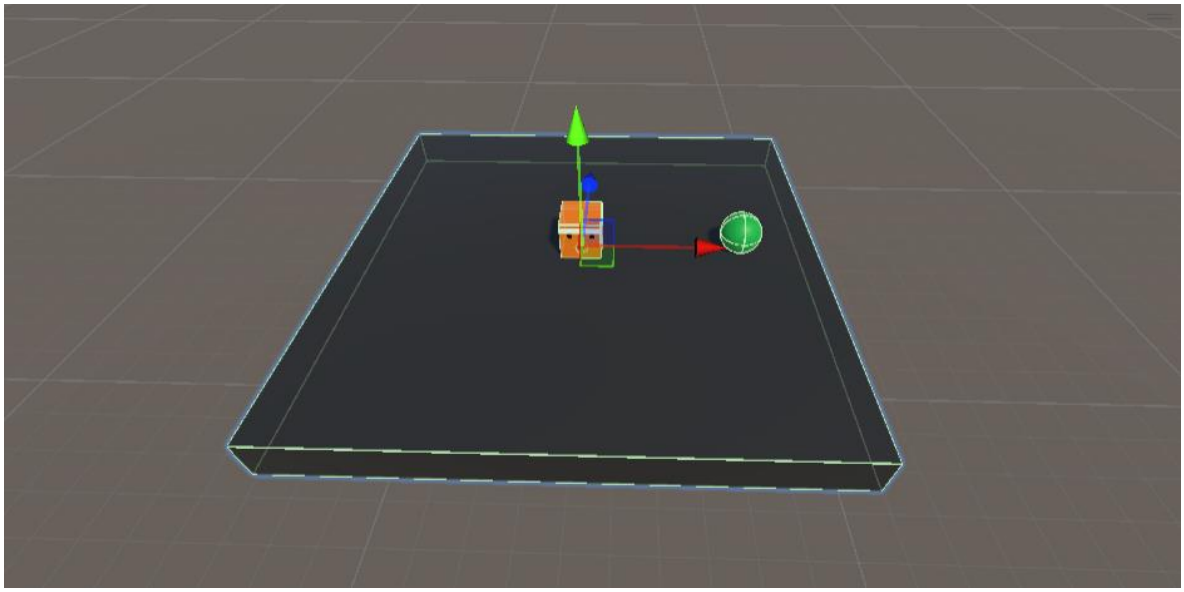


Рисунок 4 - Сцена с тривиальным поведением агента

Для обучения агента необходимо написать скрипт, где необходимо наследовать класс Agent и переопределить три метода: OnEpisodeBegin, CollectObservations, OnActionReceived [40]. OnEpisodeBegin отвечает за настройку параметров и сцены перед следующим эпизодом обучения. В нашем случае положение агента обнуляется до базового состояния и положение награды задается случайным образом из определенного диапазона (рисунок 5).

```
public override void OnEpisodeBegin()
{
    transform.localPosition = new Vector3(0, 1, 0);
    transform.localRotation = Quaternion.identity;
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;

    int x = Random.Range(-4, 4);
    int z = Random.Range(-4, 4);

    if (x == 0 && z == 0)
    {
        x = 4;
        z = 4;
    }

    targetTransform.localPosition = new Vector3(x, 1, z);
}
```

Рисунок 5 - Обновление сцены перед следующим эпизодом

Метод CollectObservations необходим для наблюдения за средой, чтобы агент мог отслеживать различные переменные сцены, которые потом отправляются в нейронную сеть в виде вектора признаков. Данный агент имеет 6 наблюдений, которые включают в себя положение агента и положение награды на сцене (рисунок 6).

```
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(transform.localPosition);
    sensor.AddObservation(targetTransform.localPosition);
}
```

Рисунок 6 - Добавление параметров среды для отслеживания

Последний метод OnActionReceived реализует различные действия агента. В метод поступает определенный сигнал, который определяет

соответствующее действие. Так, агент в нашей сцене имеет возможность перемещаться по двум координатам (рисунок 7–8).

```
public override void OnActionReceived(ActionBuffers actions)
{
    MoveAgent(actions.DiscreteActions);

    if (transform.localPosition.y < 0.5f)
    {
        EndEpisode();
        StartCoroutine(GroundMaterial(loseMaterial, 0.2f));
    }

    AddReward(-1f / MaxStep);
}
```

Рисунок 7 - Реализация действий агента

```
public void MoveAgent(ActionSegment<int> act)
{
    var dirToGo = Vector3.zero;
    var rotateDir = Vector3.zero;
    var action = act[0];

    switch (action)
    {
        case 1:
            dirToGo = transform.right * 1f;
            break;
        case 2:
            dirToGo = transform.right * -1f;
            break;
        case 3:
            rotateDir = transform.up * 1f;
            break;
        case 4:
            rotateDir = transform.up * -1f;
            break;
        case 5:
            dirToGo = transform.forward * -0.75f;
            break;
        case 6:
            dirToGo = transform.forward * 0.75f;
            break;
    }
    transform.Rotate(rotateDir, Time.fixedDeltaTime * 200f);
    rb.AddForce(dirToGo * agentRunSpeed, ForceMode.VelocityChange);
}
```

Рисунок 8 - Перемещение агента

На рисунке 9 показана реализация смены материала пола, которая служит для визуального отображения успешности выполнения агентом действий во время процесса обучения.

```
ссылок: 2
IEnumerator GroundMaterial(Material mat, float time)
{
    groundMeshRenderer.material = mat;
    yield return new WaitForSeconds(time);
    groundMeshRenderer.material = groundMaterial;
}
```

Рисунок 9 - Смена материала у пола на сцене

Существует еще метод Heuristic, который нужен для того, чтобы была возможность управления агентом самостоятельно. Как правило, таким образом тестируют сцену на корректность работы (рисунок 10).

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    if (Input.GetKey(KeyCode.D))
    {
        discreteActionsOut[0] = 3;
    }
    else if (Input.GetKey(KeyCode.W))
    {
        discreteActionsOut[0] = 1;
    }
    else if (Input.GetKey(KeyCode.A))
    {
        discreteActionsOut[0] = 4;
    }
    else if (Input.GetKey(KeyCode.S))
    {
        discreteActionsOut[0] = 2;
    }
}
```

Рисунок 10 - Ручное управление агентом

Обучение с подкреплением требует вознаграждений, чтобы сигнализировать, какие решения хороши, а какие плохи. Алгоритм обучения использует вознаграждения, чтобы определить, дает ли он Агенту оптимальные действия. Для выдачи награды агенту используется метод SetReward. Агент получит вознаграждение равное 1 только в одном случае, когда заберет награду на сцене. Если он выйдет за пределы допускаемой области, то получит награду -1. Также присутствует постоянная награда -1/MaxStep, которая должна стимулировать агента быстрее искать решение.

Агент включает в себя базовые компоненты, а также несколько определенных, которые необходимы для обучения. Они представлены на рисунке 11.

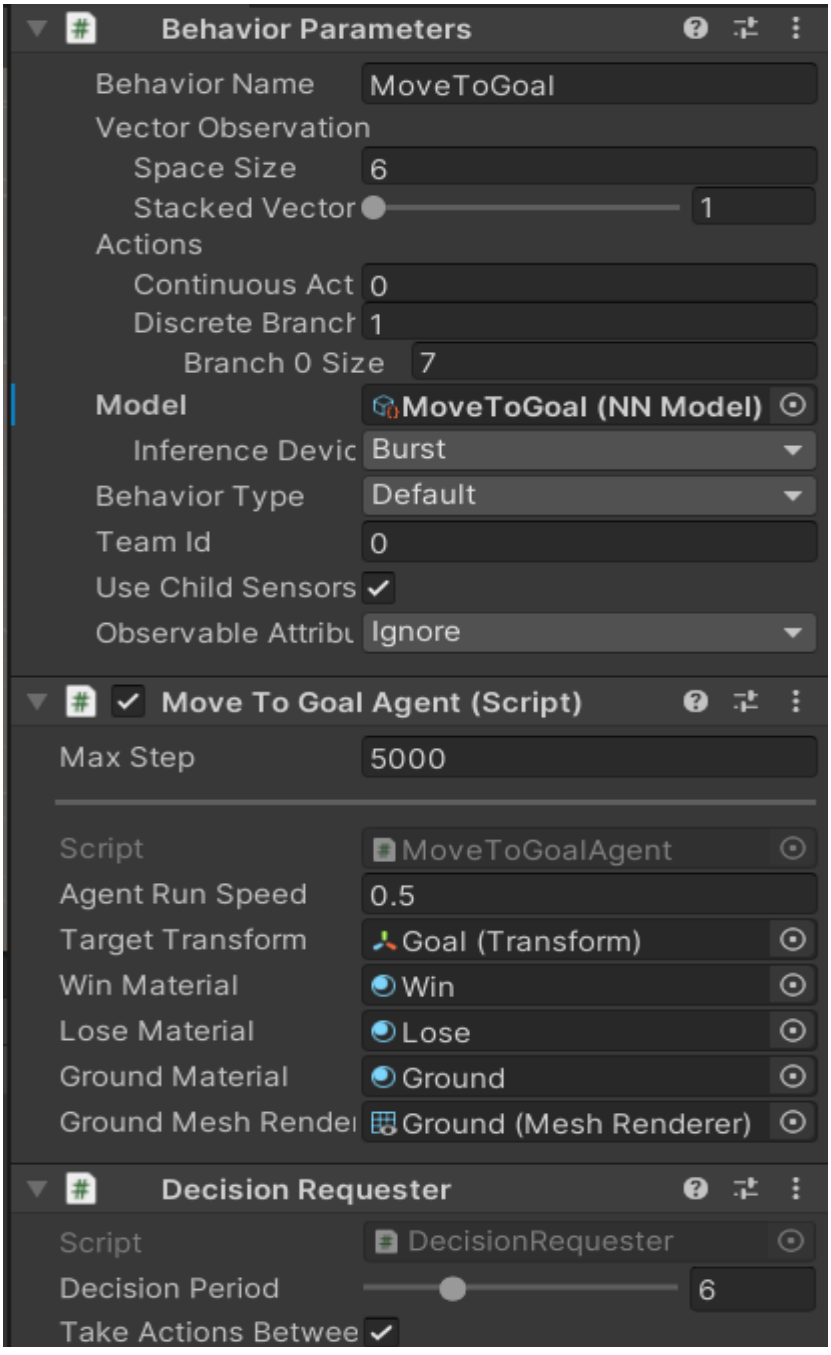


Рисунок 11 - Необходимые компоненты для обучения

Компонент Behavior Name отвечает за настройки поведения, в нем настраиваются количество наблюдений и действий агента, есть возможность

задать готовую обученную модель и другие параметры. MoveToGoal является собственным скриптом для поведения агента, у которого настраиваются следующие поля: скорость агента, положение награды, различные материалы, которые нужны для визуального отображения во время обучения. Компонент Decision Requester позволяет агенту самостоятельно запрашивать решения через регулярные промежутки времени, не используя вызов определенного метода. На рисунке 12 показана схема обучения агента.

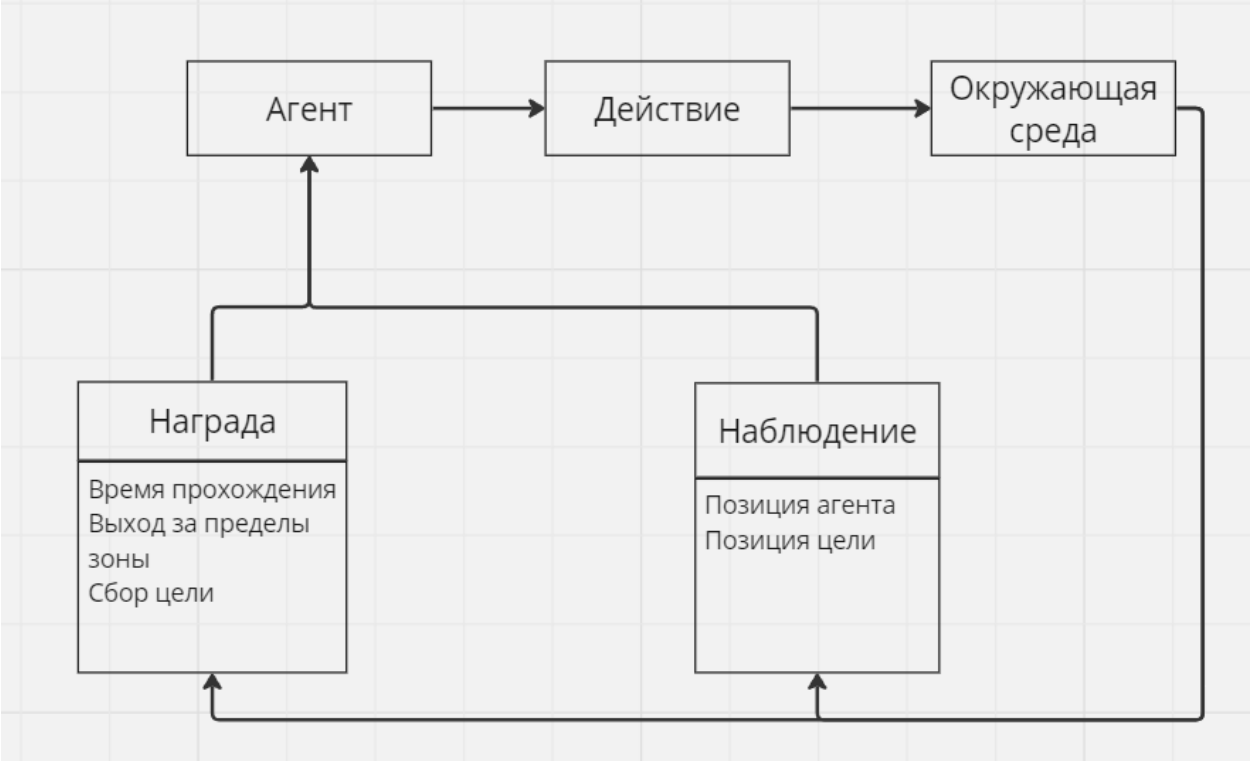


Рисунок 12 - Схема обучения агента

На рисунке 13 изображен файл конфигурации с гиперпараметрами для обучения модели.

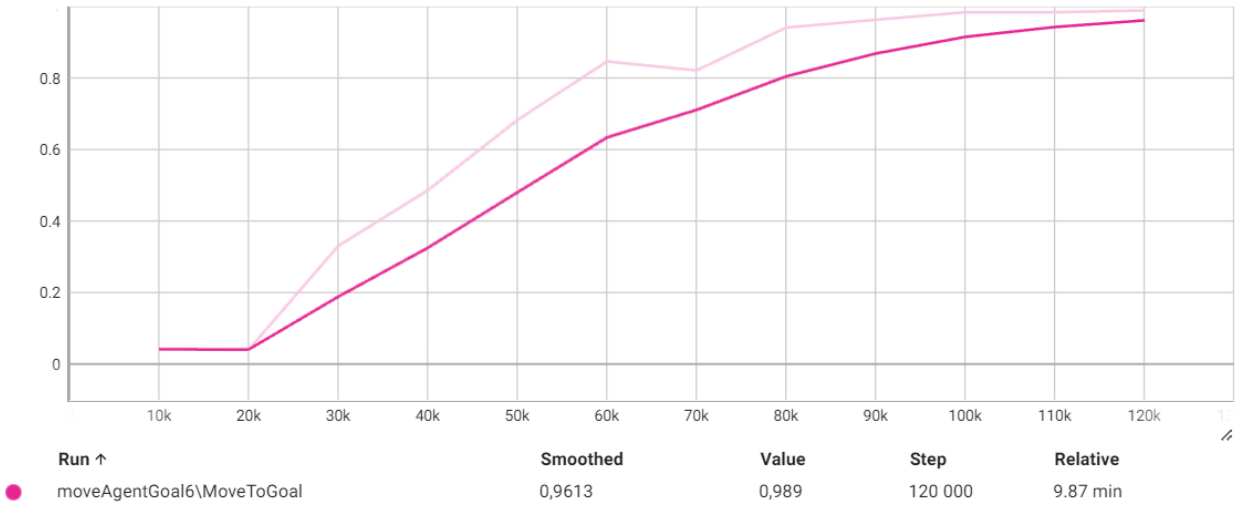
```
behaviors:
  MoveToGoal:
    trainer_type: ppo
    hyperparameters:
      batch_size: 32
      buffer_size: 128
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000
    time_horizon: 64
    summary_freq: 10000
```

Рисунок 13 - Файл конфигурации для обучения модели

В данном файле сначала следует имя обучаемого агента, а затем параметры его обучения. Агент будет обучаться по ppo алгоритму, с количеством шагов 500000, с количеством модулей в нейронной сети равным 128, в которой 2 скрытых слоя, с настройкой внешнего вознаграждения по умолчания.

После процесса обучения можно посмотреть статистику с помощью графиков tensorboard. Есть три типа графиков: статистика среды, статистика политики и функции потерь обучения [40].

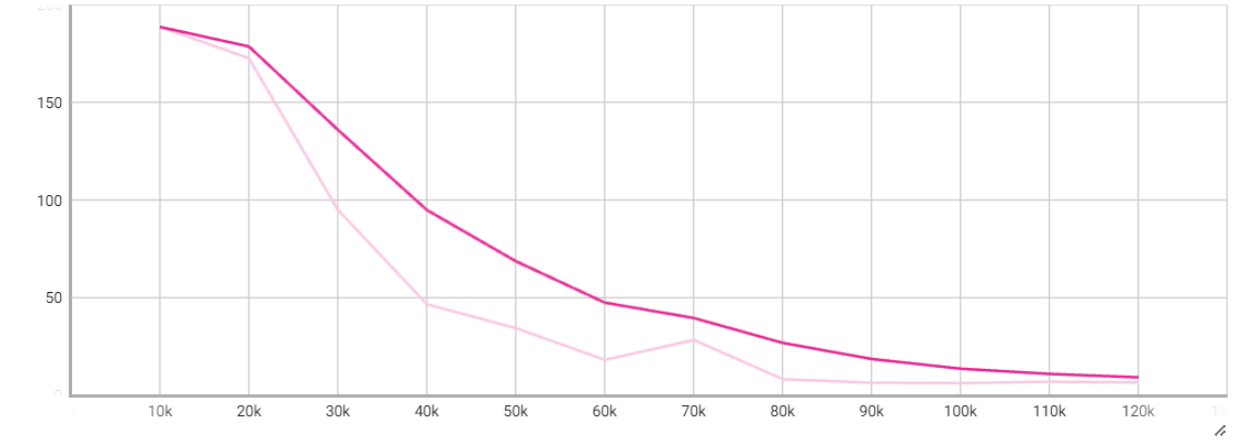
На рисунке 14 изображено среднее совокупное вознаграждение за эпизод для агента. Если обучение успешно, то значение должно постепенно увеличиваться.



Run ↑	Smoothed	Value	Step	Relative
moveAgentGoal6\MoveToGoal	0,9613	0,989	120 000	9.87 min

Рисунок 14 - Среднее совокупное вознаграждение за эпизод для агента

На рисунке 15 изображена средняя продолжительность каждого эпизода в среде для агента.



Run ↑	Smoothed	Value	Step	Relative
moveAgentGoal6\MoveToGoal	9,7897	7,1835	120 000	9.87 min

Рисунок 15 - Средняя продолжительность эпизода

На рисунке 16 изображена статистика решений модели, насколько они случайны. График медленно уменьшается в процессе успешного обучения. Если значение уменьшается слишком быстро, то необходимо увеличить beta гиперпараметр.

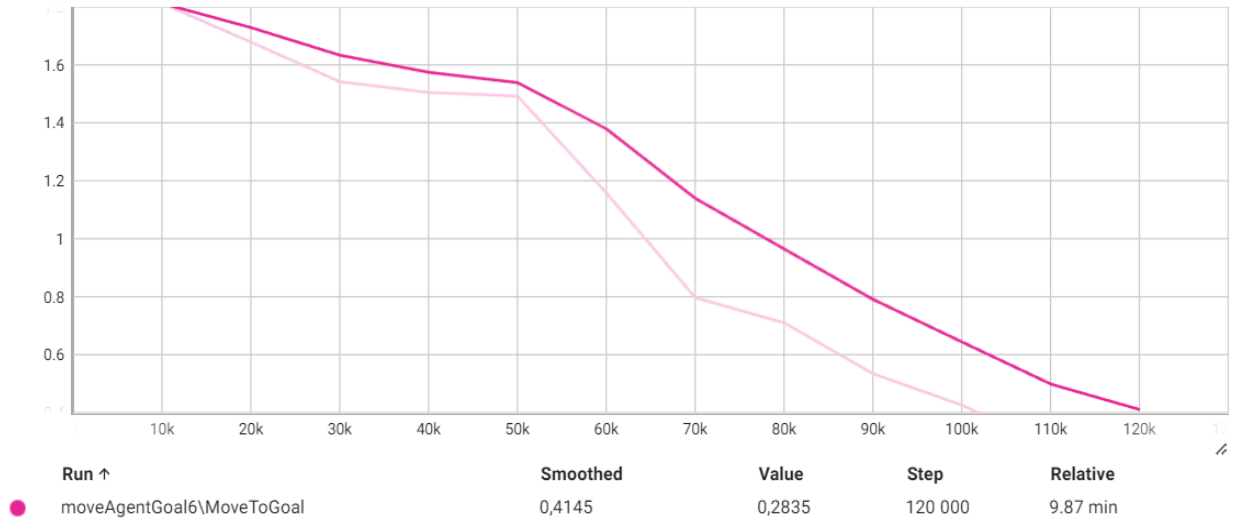


Рисунок 16 – Случайность решений модели

На рисунке 17 изображен шаг выполнения алгоритма обучения при поиске оптимальной политики. Со временем обучения шаг должен уменьшаться.

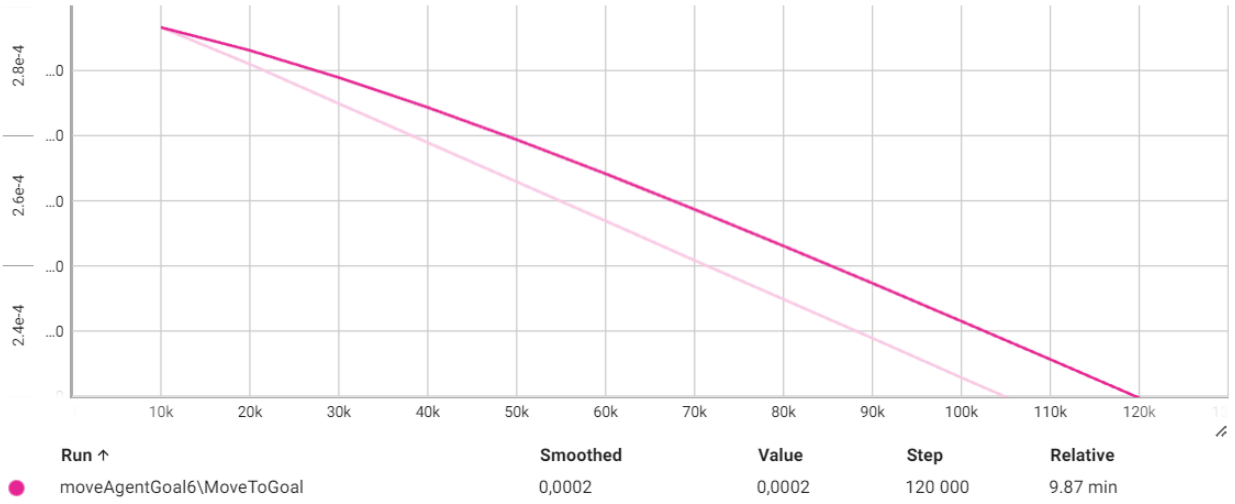


Рисунок 17 - Скорость обучения при поиске оптимальной политики

На рисунке 18 изображена средняя величина функции потерь политики. Коррелирует с тем, насколько меняется политика. Величина данного значения должна уменьшаться во время успешной сессии обучения.

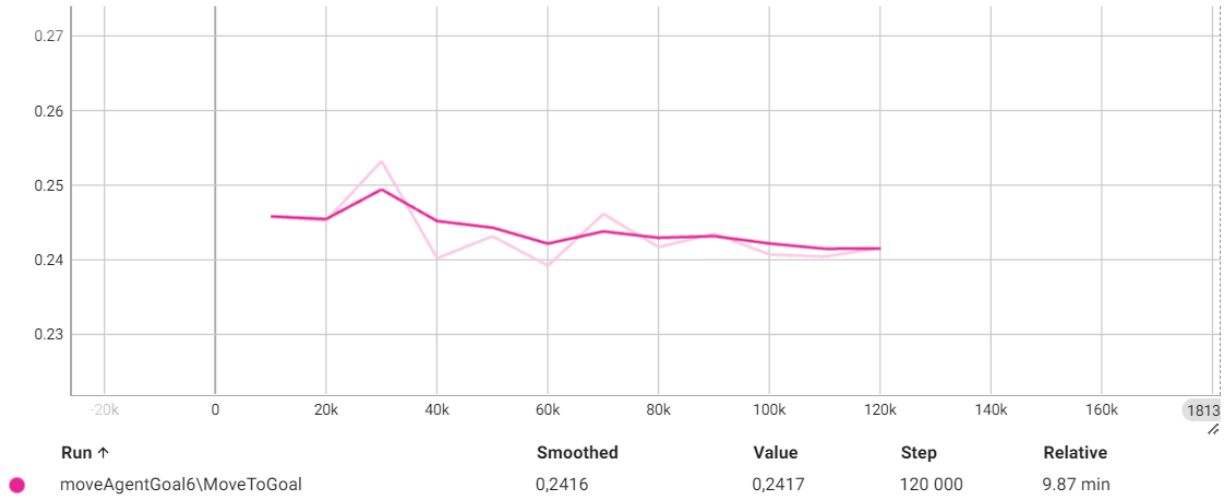


Рисунок 18 - Средняя величина функции потерь политики

На рисунке 19 отображена средняя потеря функции обновления значения. Коррелирует с тем, насколько хорошо модель способна предсказывать значение каждого состояния. Это значение должно увеличиваться, пока агент обучается, а затем уменьшаться, как только вознаграждение стабилизируется.

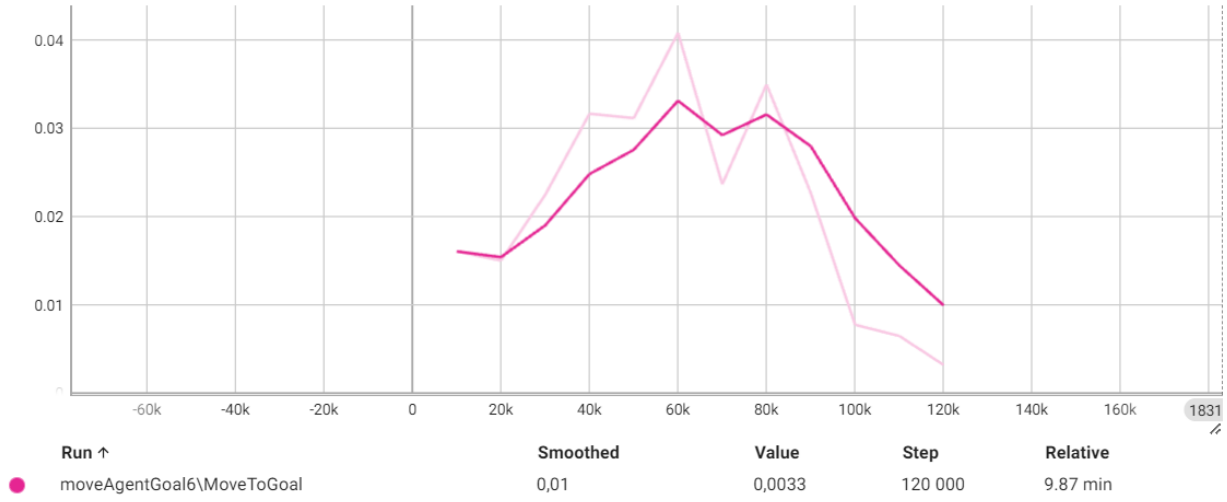


Рисунок 19 - Средняя потеря функции обновления значения

По результатам обучения и по собранным графикам видно, что агент успешно обучился со своей требуемой задачей. В процессе тестирования также заметно, что он справляется с необходимыми действиями без каких-либо проблем.

3.2 Интерактивное поведение агента: манипуляция объектами для достижения целей

С одной простой задачей обучение агента прошло без трудностей. Теперь попробуем усложнить поведение агента. В следующей сцене цель агента также заключается в том, чтобы собрать награду, но теперь для начала ему необходимо переместить блок в определенную зону, которая активирует появление награды для агента. На рисунке 20 изображена данная сцена.

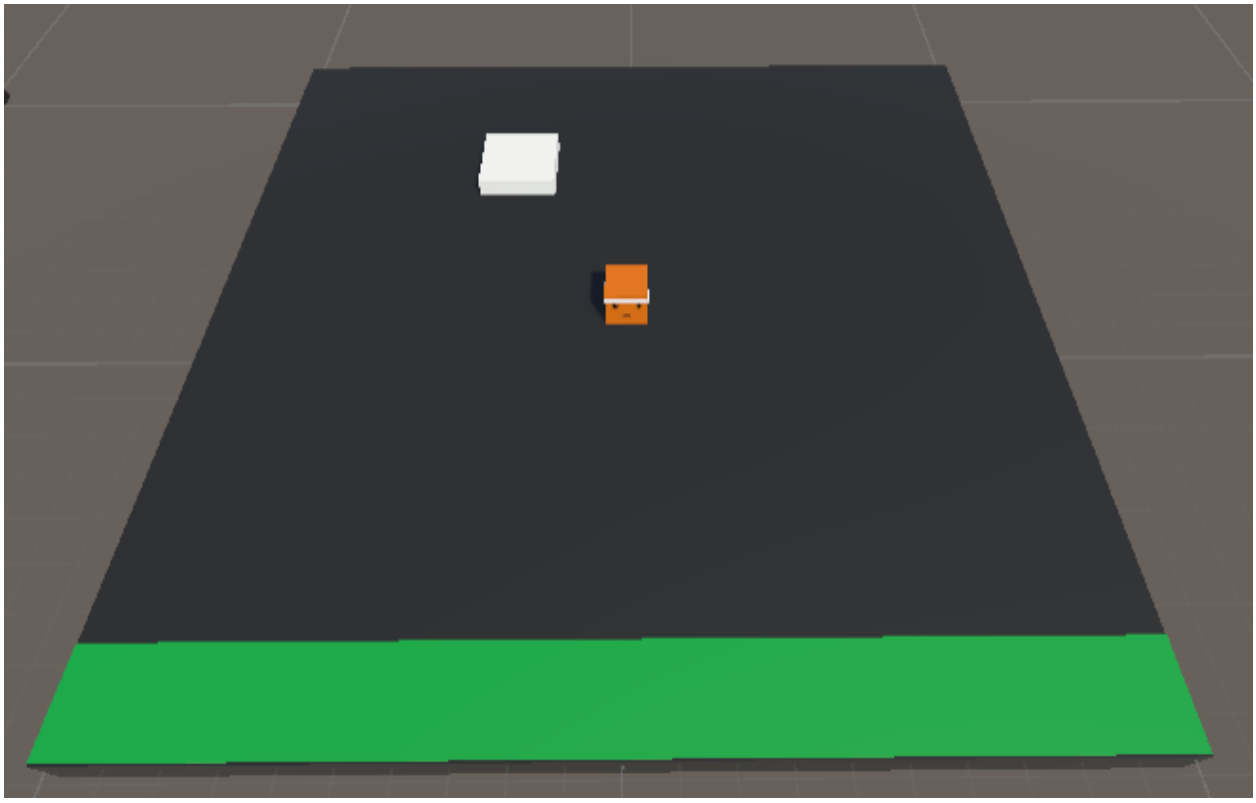


Рисунок 20 - Сцена с интерактивным поведением агента

Данный агент копирует сценарий поведения агента из предыдущей сцены, но с дополнительными условиями. Так, добавлен код для создания награды на сцене и код для проверки текущей награды (рисунок 21).

```
public void SpawnGoal()
{
    goal = Instantiate(target, area.transform);
    goal.transform.localPosition = new Vector3(4, 1, 0);
}

Ссылок: 1
bool HasGoal()
{
    return goal != null;
}
```

Рисунок 21 - Создание и проверка на наличие награды

Так же на сцену были добавлены дополнительные ограничения в виде невидимых стен для того, чтобы ускорить процесс обучения, так как агент в процессе обучения часто проваливался и сбрасывал необходимый блок. Поэтому был написан код для распознавания столкновений со стенами (рисунок 22).

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("wall"))
    {
        StartCoroutine(GroundMaterial(loseMaterial, 0.2f));
        EndEpisode();
    }
}
```

Рисунок 22 - Проверка на коллизии

На рисунке 23 изображены основные компоненты агента с заданными параметрами. Скрипт агента повторяет предыдущий, но добавляет две переменные связанные с расположением блока и зоны с активацией.

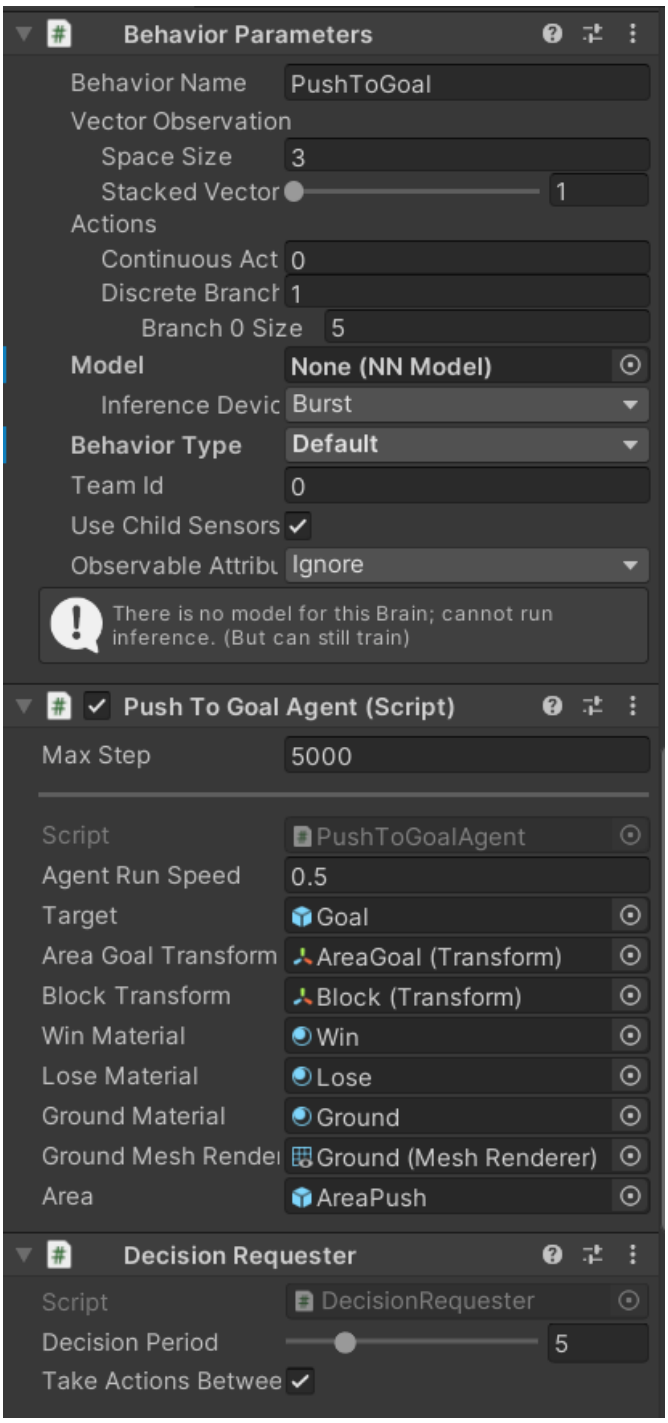


Рисунок 23- Основные компоненты

Агент содержит только наблюдения за своим расположением, так как остальную информацию о сцене он получает, используя лучи, за которые отвечает компонент Ray Perception Sensor 3D [41]. На рисунке 24 отображена работа данного компонента. В его настройках указываются различные параметры лучей, а также на какие объекты лучи должны реагировать.

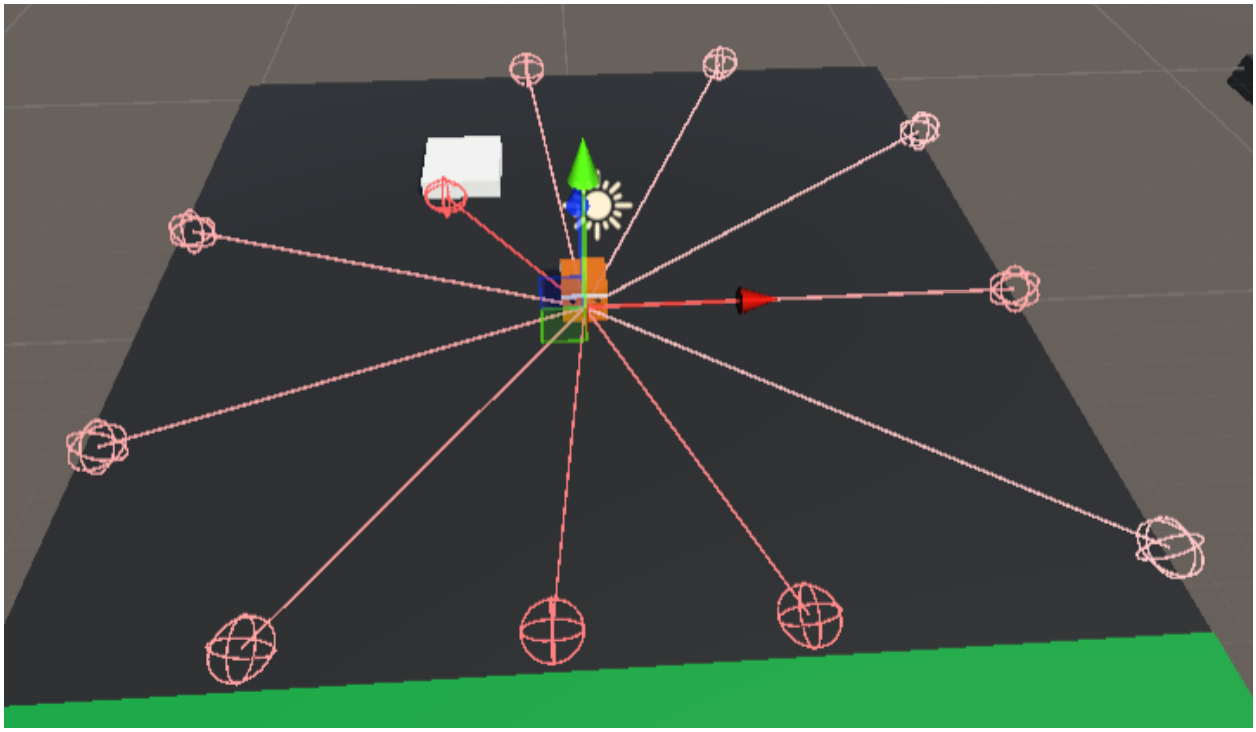


Рисунок 24 - Лучи агента

На рисунке 25 изображен конфигурационный файл. Основная часть была взята с файла из прошлой сцены, но с небольшими изменениями.

```
behaviors:
  PushToGoal:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 1024
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
      gail:
        gamma: 0.99
        strength: 0.01
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
        learning_rate: 0.0003
        use_actions: false
        use_vail: false
        demo_path: Demos/PushToGoalDemo.demo
    keep_checkpoints: 5
    max_steps: 400000
    time_horizon: 128
    summary_freq: 20000
    behavioral_cloning:
      demo_path: Demos/PushToGoalDemo.demo
      strength: 1.0
      steps: 200000
      samles_per_update: 0
```

Рисунок 25 - Файл конфигурации

Также есть два параметра, которые играют важную роль. Unity предоставляет возможность записать демонстрацию поведения, показывая модели, как должен действовать агент [42]. Оба параметра отвечают за то, насколько модель будет соответствовать поведению из демонстрации. Gail хорошо работает в совокупности с внешними вознаграждениями в отличии от behavioral_cloning. В нашем случае behavioral_cloning нужен для того, чтобы модель смогла понять основные действия агента для получения награды, после половины обучения клонирование отключается и агент обучается посредством

внешних и внутренних наград [43]. Схема обучения модели показана на рисунке 26.

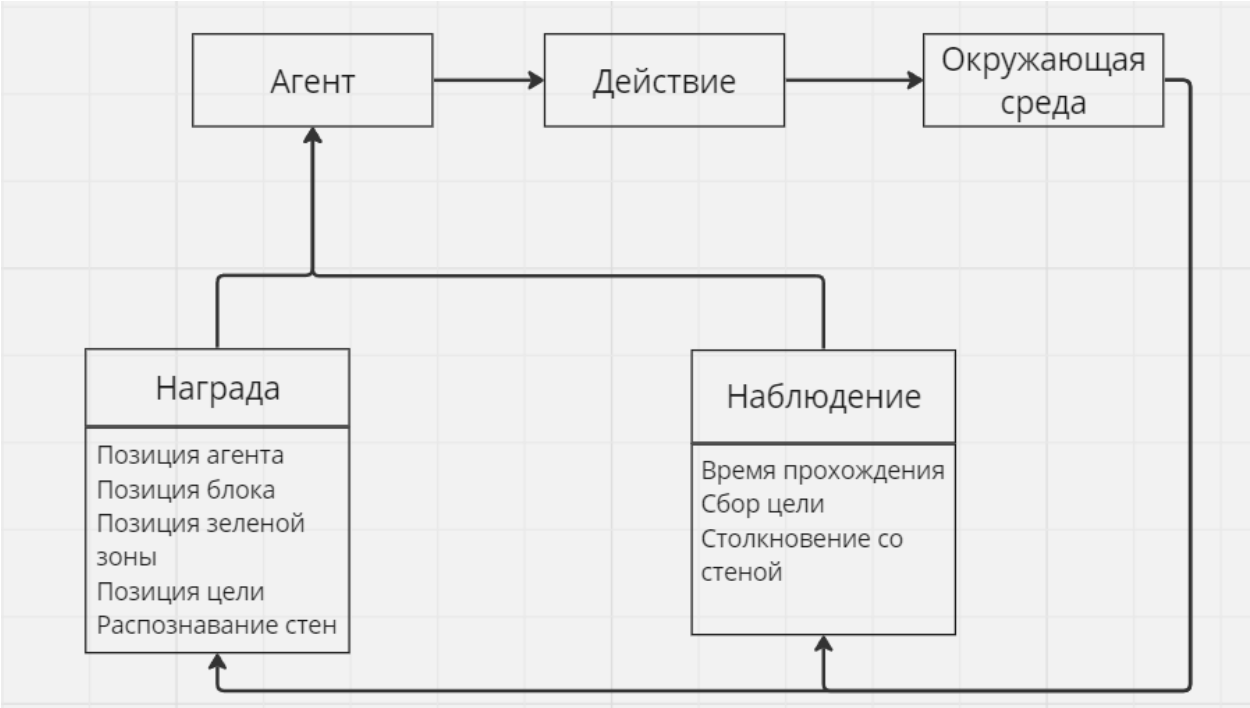


Рисунок 26 – Схема обучения агента

На рисунках 27–30 изображена статистика обучения агента.

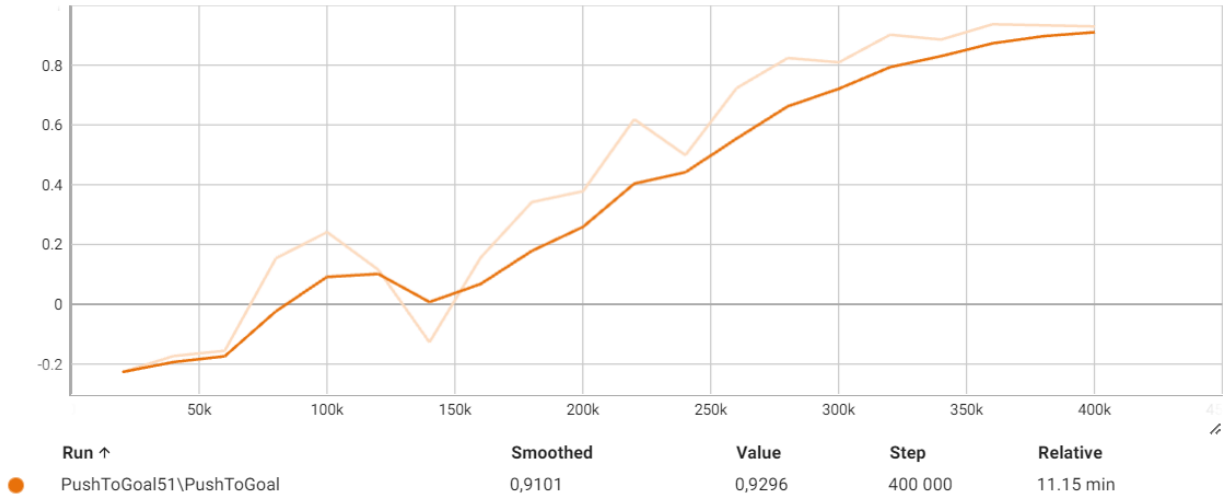


Рисунок 27 - Среднее совокупное вознаграждение за эпизод для агента

Средняя величина потерь дискриминатора GAIL. Соответствует тому, насколько хорошо модель имитирует демонстрационные данные.

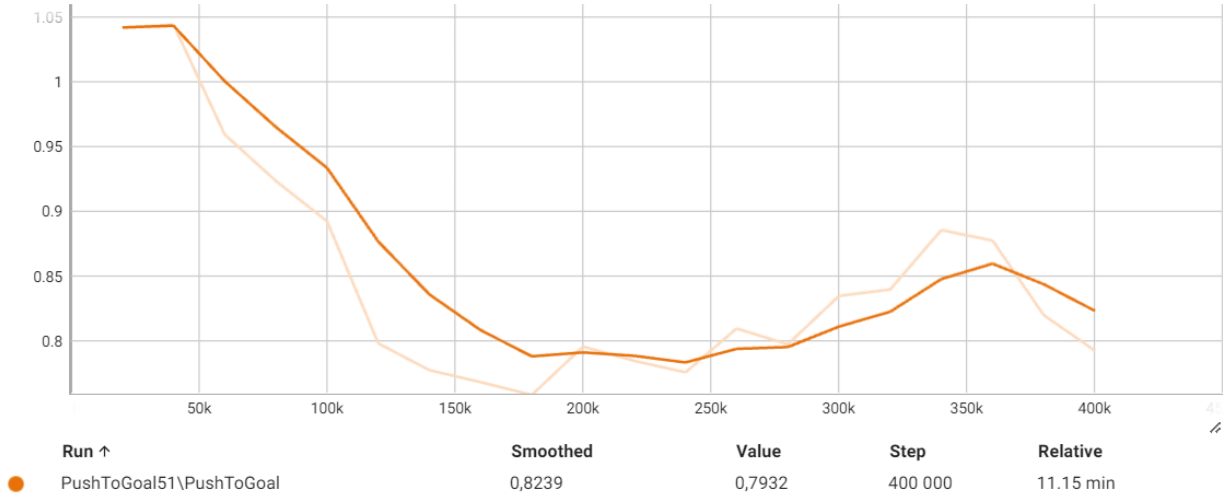


Рисунок 28 - Средняя величина потерь дискриминатора GAIL

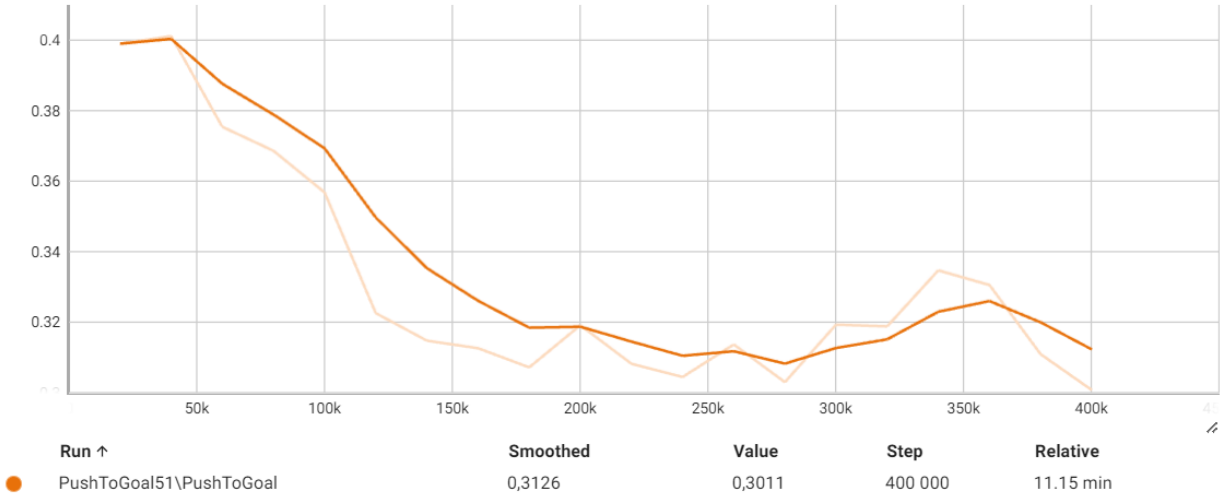


Рисунок 29 - Оценка дискриминатора для состояний и действий, генерируемых политикой

Средняя величина потери поведенческого клонирования. Соответствует тому, насколько хорошо модель имитирует демонстрационные данные.

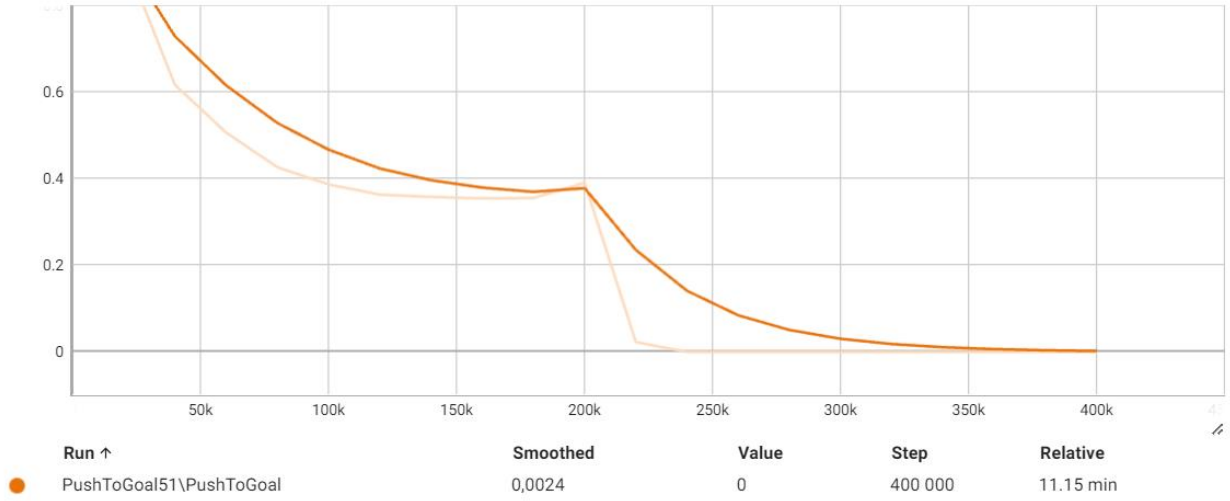


Рисунок 30 - Средняя величина потери поведенческого клонирования

После обучения агента видно, что процесс завершился успешно. Также по графикам можно отследить информацию по уровню имитации агента демонстрационным данным. Отчетливо заметно, что после 200 тысяч эпизодов агент стал хуже имитировать эти данные, но стал больше полагаться на внешние и внутренние вознаграждения.

3.3 Взаимодействие агентов с разными ролями

В третьей сцене реализуется взаимодействие двух агентов с различными ролями: один агент играет роль собирателя наград, а другой — роль охотника (рисунок 31). Целевой агент должен перемещаться по сцене и собирать награды, избегая при этом охотника [44]. Охотник, в свою очередь, должен поймать целевого агента, чтобы завершить свою задачу. На рисунке изображена данная сцена. Оба агента получают награду равную -15 за невыполнение необходимых действий. Агент в роли сборщика получает вознаграждение за каждую собранную награду в виде +5 и за все +10, если агента ловят, то он получает -10

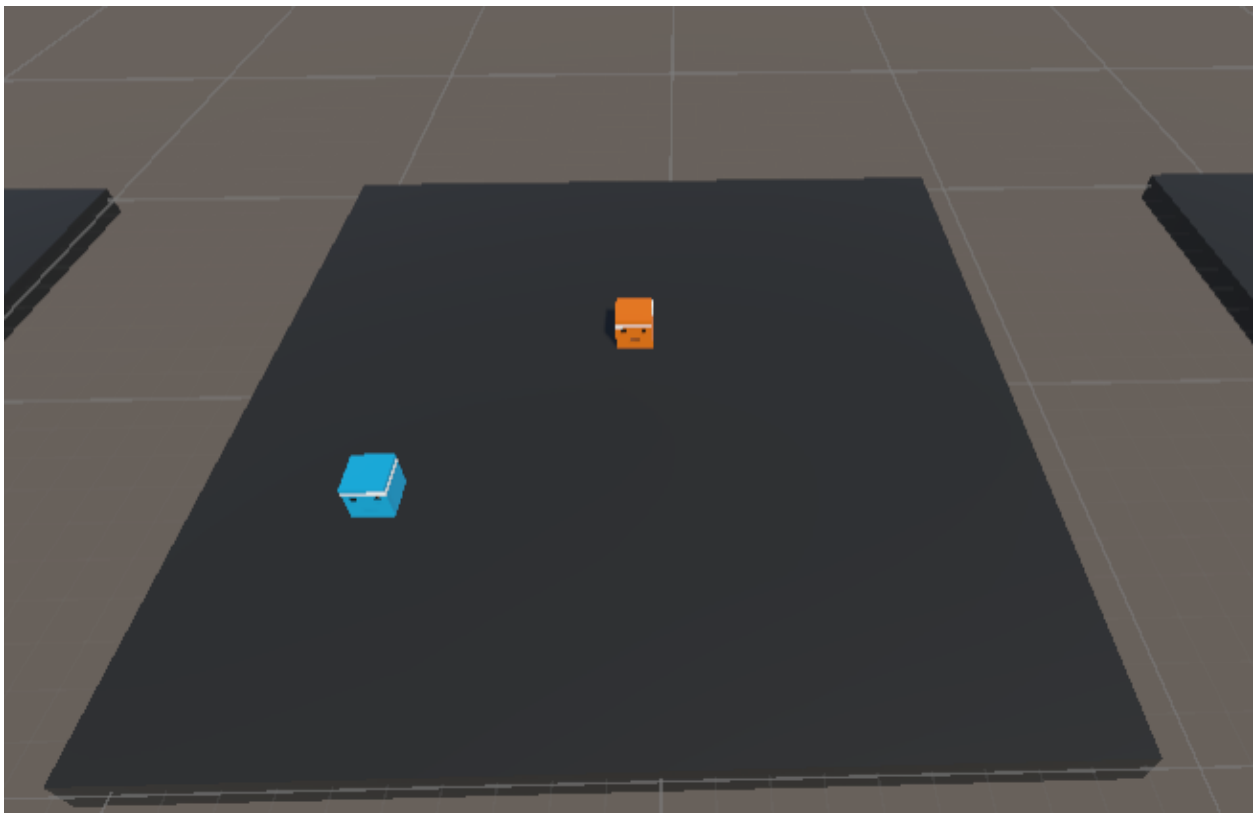


Рисунок 31 - Взаимодействие агентов с разными ролями

На рисунках 32–33 отображены главные компоненты агентов. У собирающего агента в коде присутствует информация об положении агента в роли охотника, количество наград на карте, их положение, префаб награды, скорость агента и класс охотника. Второй агент содержит заданную скорость, объект жертвы и его класс.

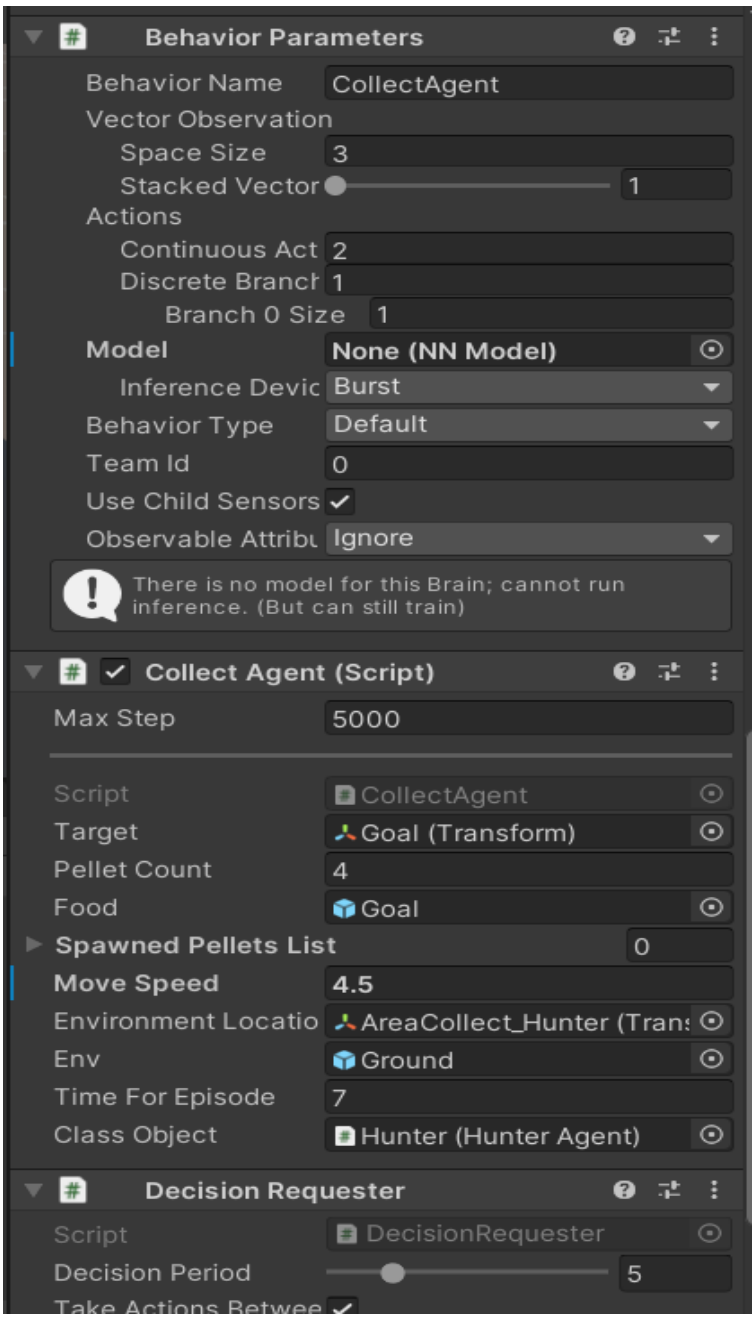


Рисунок 32 - Основные компоненты целевого агента

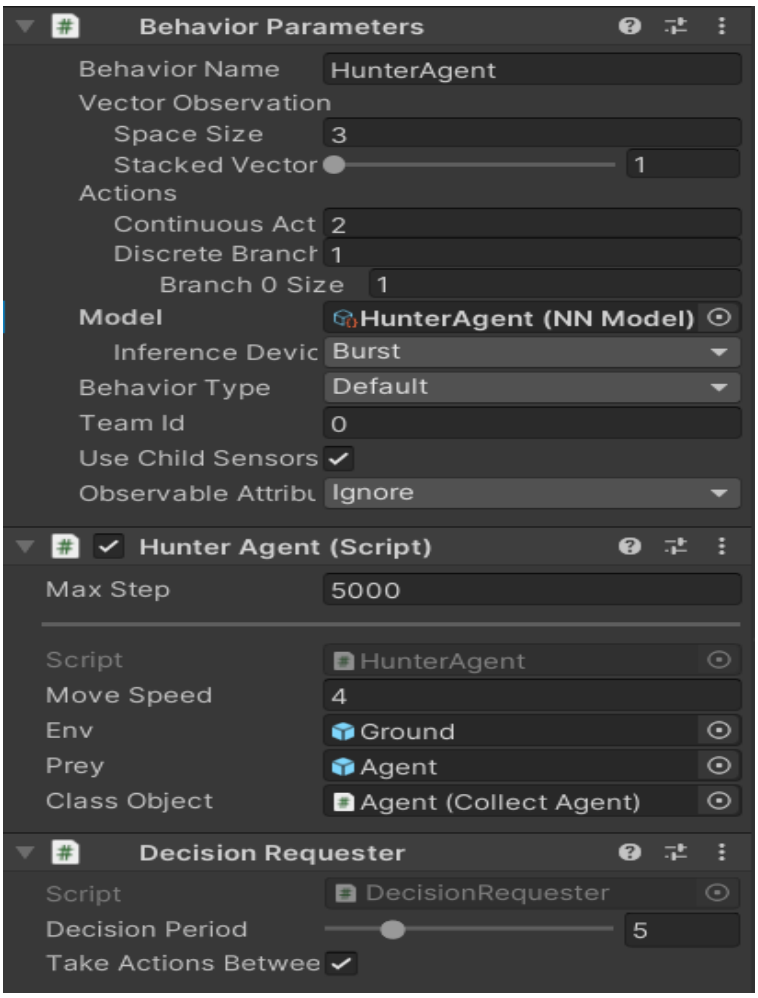


Рисунок 33 - Основные компоненты преследующего агента

Оба агента имеют компонент лучей, который показан на рисунке 34.

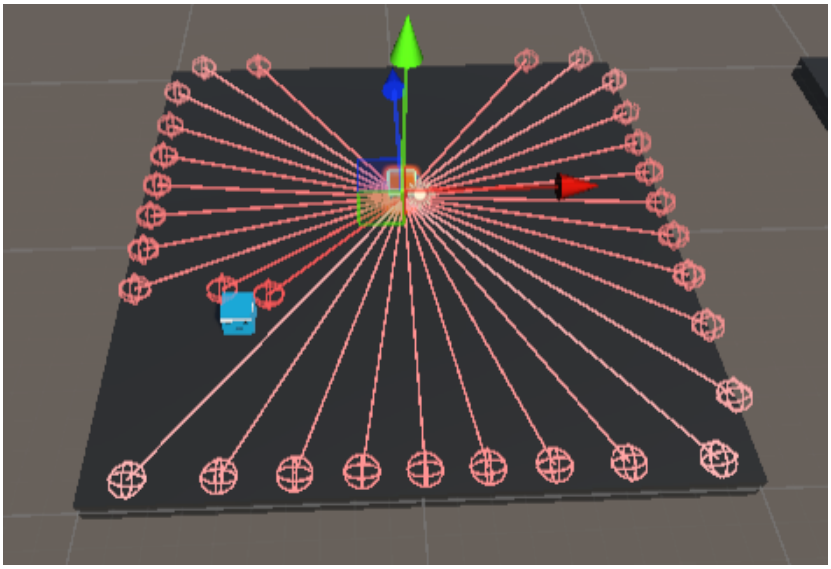


Рисунок 34 - Отображение лучей целевого агента

Их настройка изображена на рисунках 35–36. Количество лучей у агентов равно 15 с длиной 20. У первого агента есть распознавание стен, охотника и наград. А у второго такое же распознавание, кроме награды.

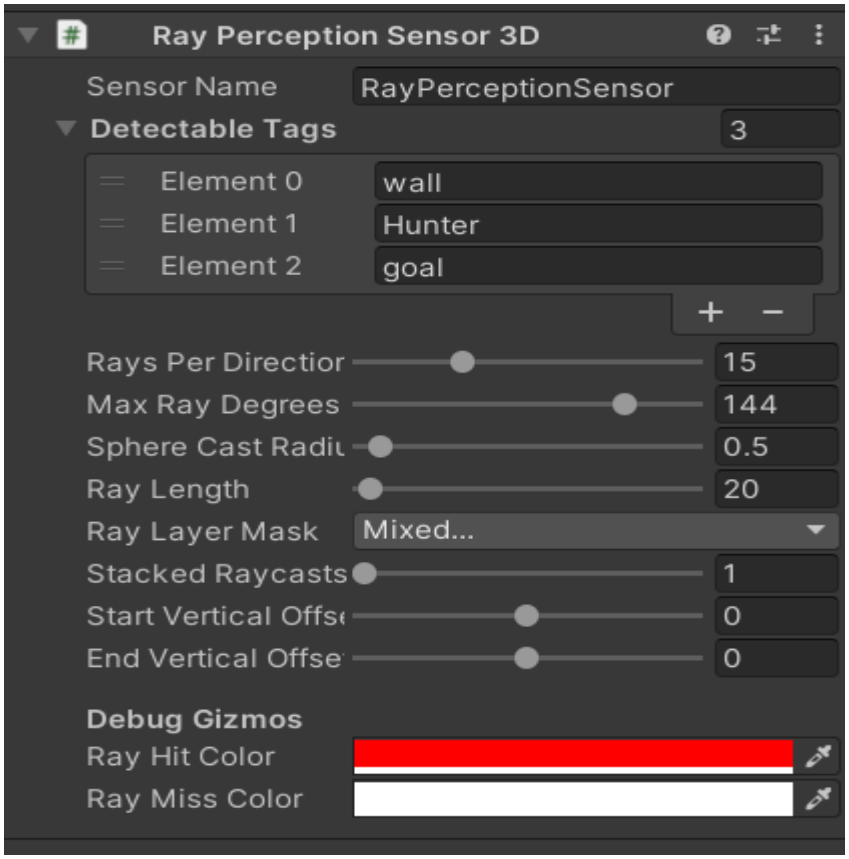


Рисунок 35 - Настройка лучей целевого агента

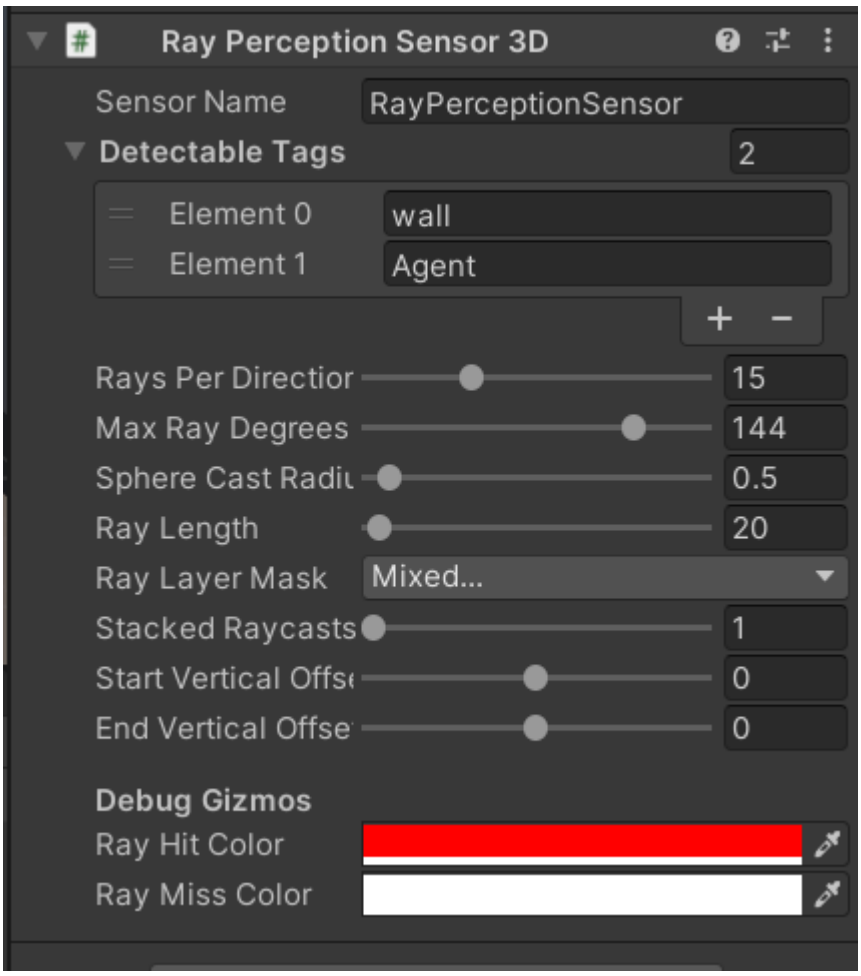


Рисунок 36 - Настройка лучей преследующего агента

На рисунке 37 отображен файл с настройками для моделей в данной сцене. Здесь можно наблюдать за тем, что Unity позволяет производить одновременно обучение нескольких агентов, в нашем случае это CollectAgent и HunterAgent [45].

```yaml
behaviors:
  CollectAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 1
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.9
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 2000000
    time_horizon: 64
    summary_freq: 50000
  HunterAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 1
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.9
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 2000000
    time_horizon: 64
    summary_freq: 50000
```

Рисунок 37 - Файл конфигурации

На рисунке 38 отображена схема обучения агентов.

Рисунок 38 - Схема обучения агентов

Среднее совокупное вознаграждение за эпизод изображено на рисунке 39. Со временем значения должны увеличиваться, но видно то, что график агента охотника практически не изменился.

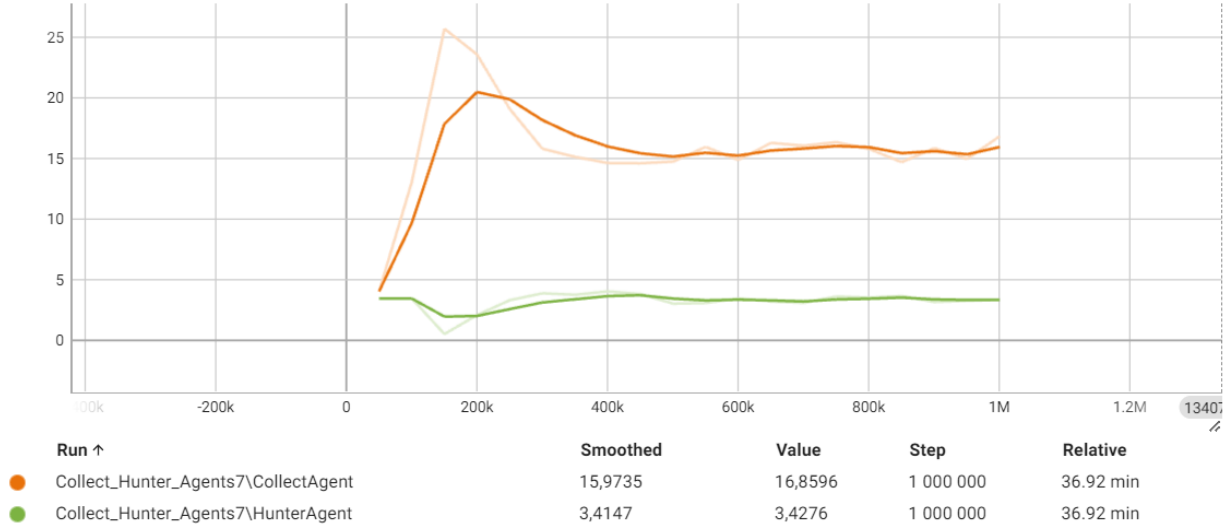


Рисунок 39 - Среднее совокупное вознаграждение за эпизод

Средняя продолжительность эпизода уменьшается, это показано на рисунке 40.

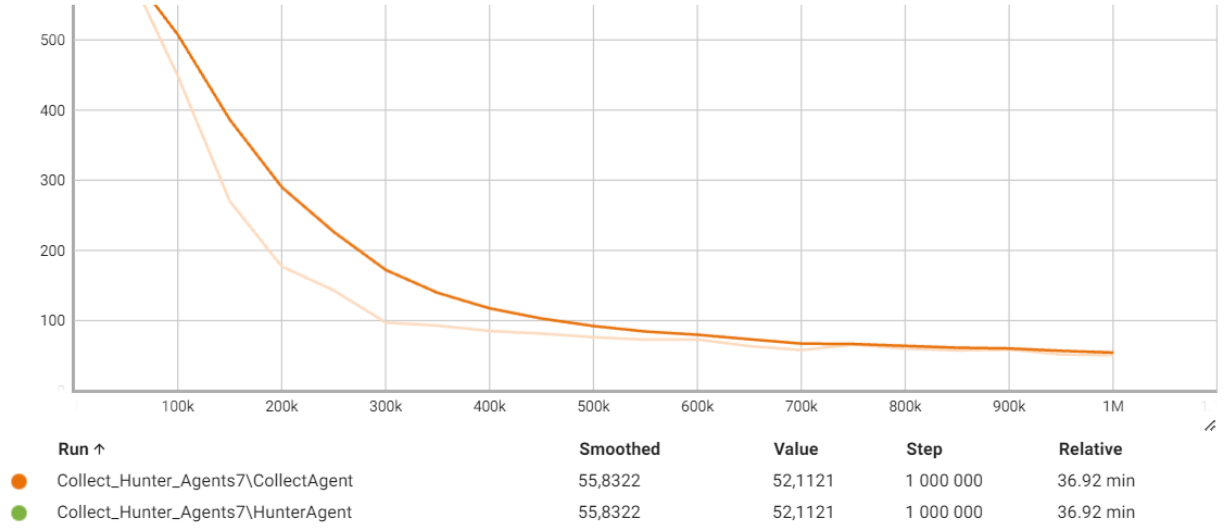


Рисунок 40 - Средняя продолжительность эпизода

На рисунке 41 изображена средняя величина функции потерь политики. Если обучение успешно, то значение должно уменьшаться. Видно, что график стабилен, правда есть один скачок, который позже стабилизируется.

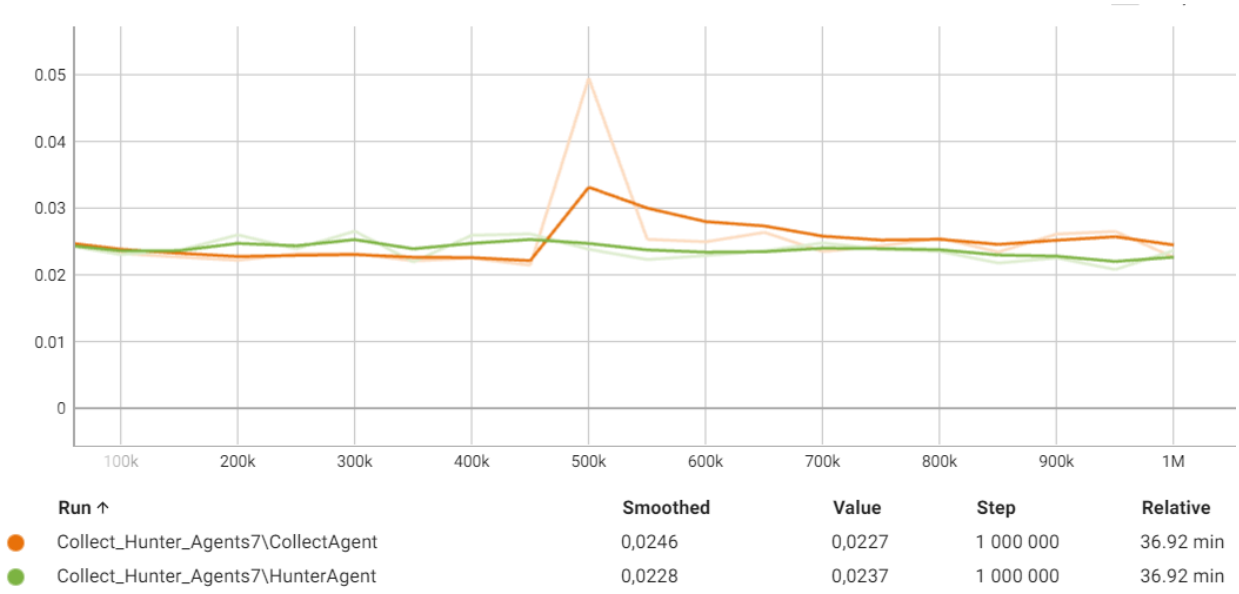


Рисунок 41 - Средняя величина функции потерь политики

Рисунок 42 показывает размер шага алгоритма, который был выбран для обучения. В данном случае обе линии возрастают, хотя должны уменьшаться.

Run ↑	Smoothed	Value	Step	Relative
Collect_Hunter_Agents7\CollectAgent	1,8623	2,0726	1 000 000	36.92 min
Collect_Hunter_Agents7\HunterAgent	0,6256	0,5918	1 000 000	36.92 min

Рисунок 42 - Скорость обучения при поиске оптимальной политики

После обучения моделей можно отметить, что агенты практически успешно справляются со своими задачами, но местами возникают определенные трудности. На графиках заметно, что есть параметры, которые необходимо скорректировать для получения лучшего результата обучения.

3.4 Сложное взаимодействие с несколькими охотниками

Следующая сцена представляет собой копию предыдущей, только в нее был добавлен дополнительный агент в роли охотника. Данная сцена изображена на рисунке 43.

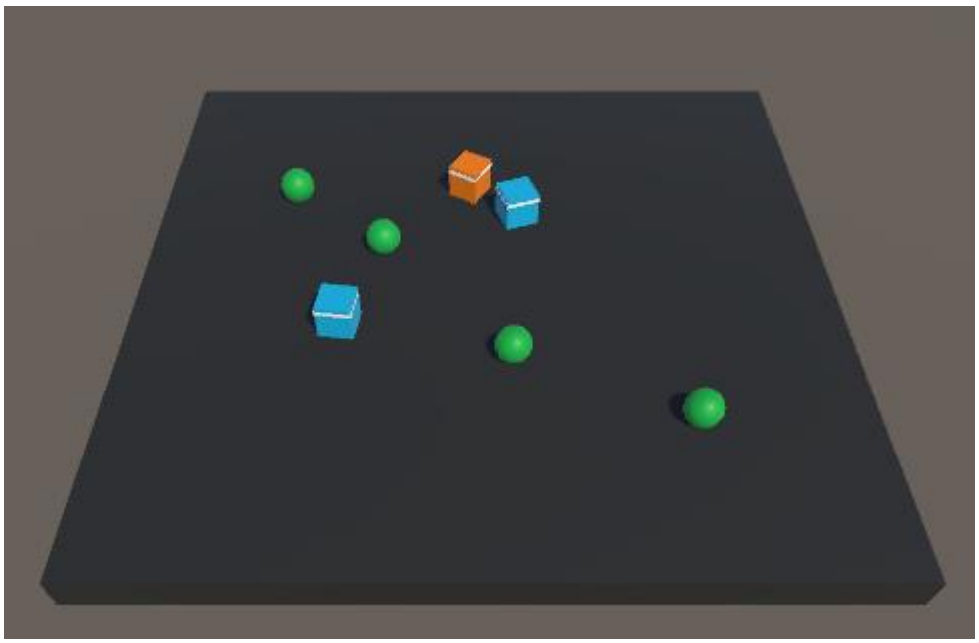


Рисунок 43 - Сложное взаимодействие с несколькими охотниками

Файл с конфигурацией для обучения остался неизменным. Среднее вознаграждение за эпизод показано на рисунке 44. Так как агентов стало больше, то их получаемая награда увеличилась, а у агента в роли сборщика она уменьшилась, так как сложность выполнения увеличилась.

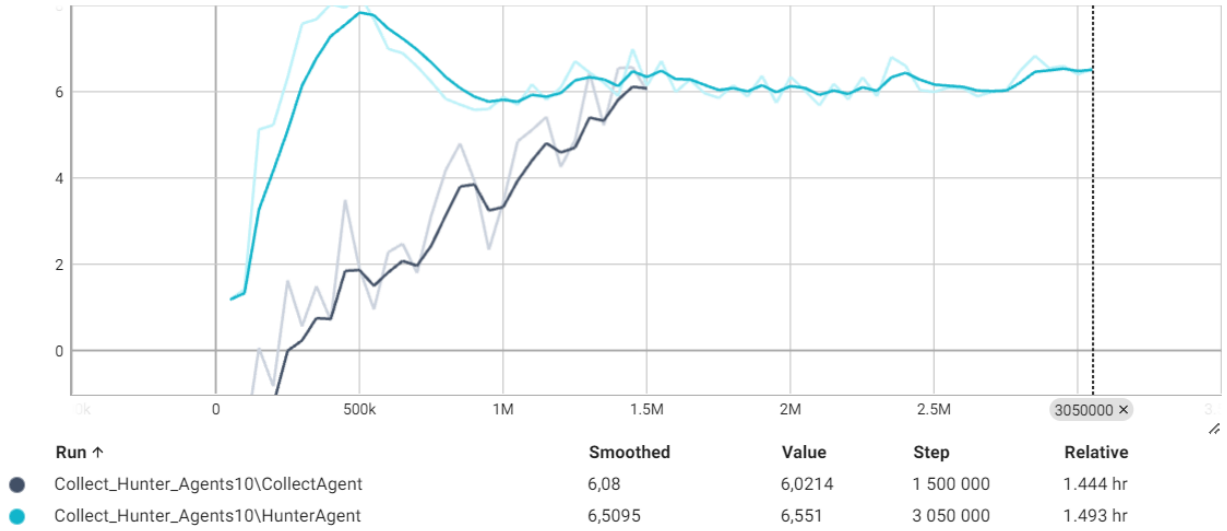
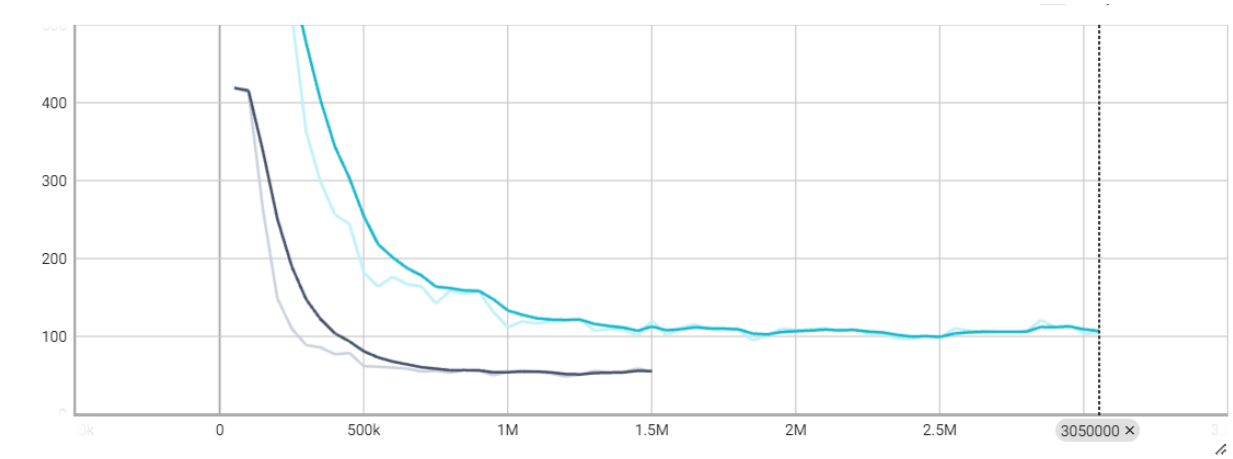


Рисунок 44 - Среднее вознаграждение за эпизод

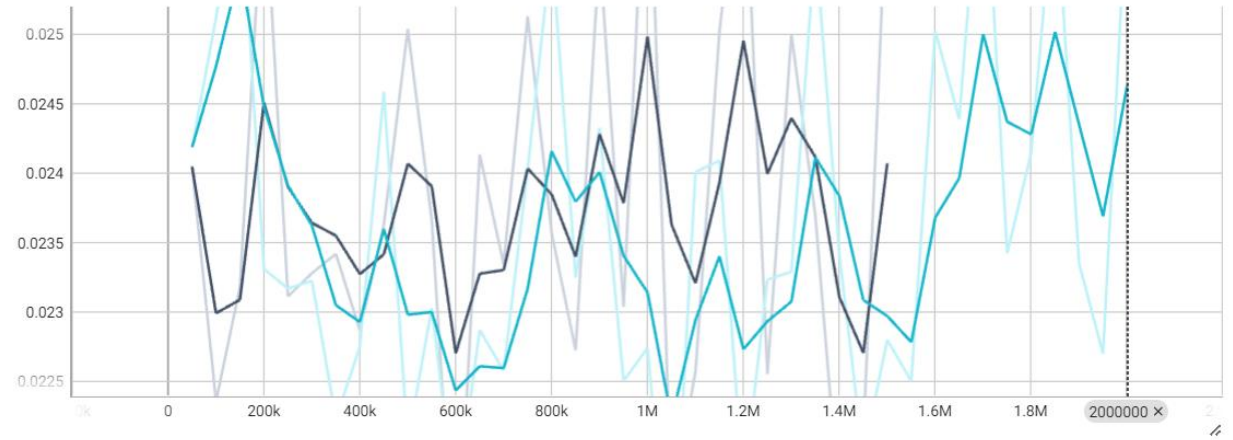
Средняя продолжительность эпизода продемонстрирована на рисунке 45. Данный график практически схож с графиком из предыдущей сцены, есть одно различие в значениях, но оно зависит от количества агентов.



Run	Smoothed	Value	Step	Relative
Collect_Hunter_Agents10\CollectAgent	56,6794	56,0912	1 500 000	1.444 hr
Collect_Hunter_Agents10\HunterAgent	108,2263	105,3861	3 050 000	1.493 hr

Рисунок 45 - Средняя продолжительность эпизода

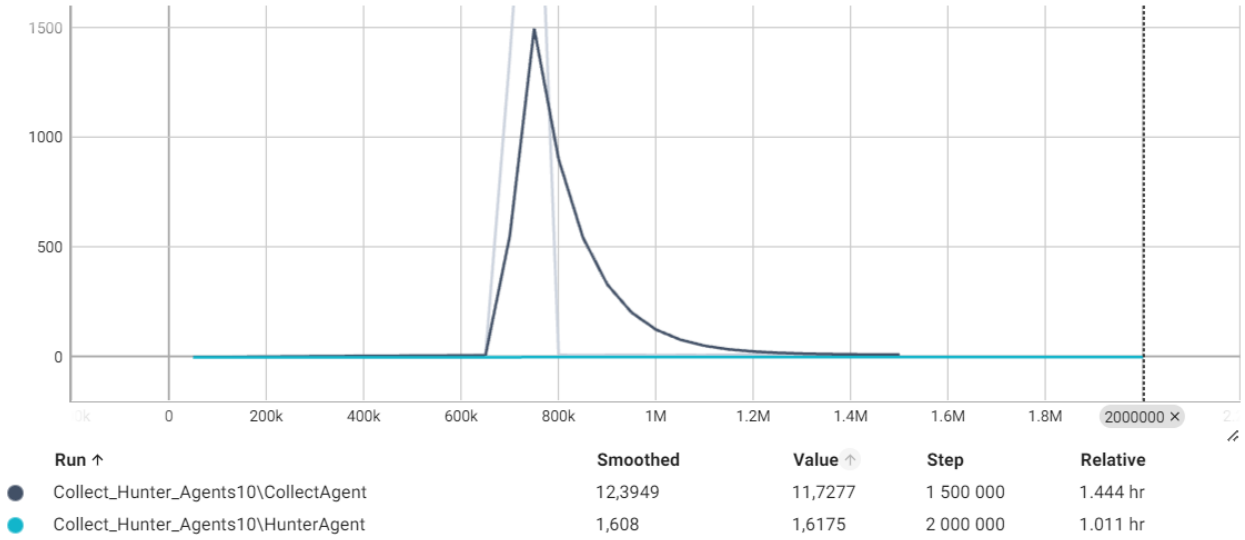
На рисунке 46 изображена средняя величина функции потерь политики. Видно, что график довольно скачкообразный, но в целом значения небольшие.



Run	Smoothed	Value	Step	Relative
Collect_Hunter_Agents10\CollectAgent	0,0241	0,0261	1 500 000	1.444 hr
Collect_Hunter_Agents10\HunterAgent	0,0246	0,026	2 000 000	1.011 hr

Рисунок 46 - Средняя величина функции потерь политики

Средняя величина функции потерь отображена на рисунке 47. Данный график также схож с графиком из прошлой сцены.



Run	Smoothed	Value	Step	Relative
Collect_Hunter_Agents10\CollectAgent	12,3949	11,7277	1 500 000	1.444 hr
Collect_Hunter_Agents10\HunterAgent	1,608	1,6175	2 000 000	1.011 hr

Рисунок 47 - Средняя величина функции потерь

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана система для обучения виртуальных союзников на базе Unity ML-Agents. Исследование включало в себя анализ существующих методов и алгоритмов машинного обучения, их интеграцию с игровой платформой Unity, а также разработку и тестирование различных сценариев поведения агентов в виртуальной среде. Основные результаты и выводы следующие:

Были рассмотрены различные способы реализации игрового ИИ, включая скриптовые методы, конечные автоматы, дерева поведения и алгоритмы машинного обучения. Проведен обзор возможностей Unity ML-Agents, что позволило выделить основные преимущества этой платформы для обучения виртуальных агентов. Изучены основные алгоритмы обучения с подкреплением, включая SARSA, PPO и Soft Actor-Critic, а также методы, основанные на искусственных нейронных сетях. Проведено сравнение этих алгоритмов, выявлены их сильные и слабые стороны в контексте обучения игровых агентов. Разработаны и протестированы четыре различных сценария, демонстрирующих разные уровни сложности.

Применение методов обучения с подкреплением, таких как PPO, показало высокую эффективность в создании адаптивных и обучающихся агентов. Агенты успешно справлялись с поставленными задачами, демонстрируя способность к обучению и адаптации к изменяющимся условиям.

На основе проведенного исследования можно выделить несколько перспективных направлений для дальнейших работ:

- Улучшение алгоритмов обучения: Исследование и применение более сложных и эффективных алгоритмов машинного обучения для повышения адаптивности и обучаемости агентов.

- Расширение сценариев: Разработка новых и более сложных игровых сценариев, включающих кооперативное и конкурентное взаимодействие агентов в большем масштабе.
- Практическое применение результатов: Разработанные методы и алгоритмы могут быть применены не только в игровой индустрии, но и в других областях, таких как робототехника, автоматизация и симуляция сложных систем. Это открывает широкие возможности для использования результатов данного исследования в различных прикладных задачах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Reinforcement Learning: Q-Algorithm in a Match to Sample Task / unrealai.wordpress.com / - Текст: электронный URL: https://unrealai.wordpress.com/2017/12/19/q-learning/ (дата обращения: 15.03.2024)

2 Learning Agents Introduction / dev.epicgames.com / - Текст: электронный URL: https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction (дата обращения: 15.03.2024)

3 MachineLearningRemote-Unreal / github.com/getnamo / - Текст: электронный URL: https://github.com/getnamo/MachineLearningRemote-Unreal (дата обращения: 15.03.2024)

4 Machine Learning in Video Games / golinuxcloud.com/ - Текст: электронный URL: https://www.golinuxcloud.com/machine-learning-in-video-games/ (дата обращения: 15.03.2024)

5 The menagerie of ways AI is transforming video game creation / atelier.net / - Текст: электронный URL: https://atelier.net/insights/ai-transforming-video-games-creation (дата обращения: 15.03.2024)

6 Machine Learning Experiments in Gaming / cloud.google.com/blog / - Текст: электронный URL: https://cloud.google.com/blog/topics/developers-practitioners/machine-learning-experiments-gaming-and-why-it-matters (дата обращения: 15.03.2024)

7 How Machine Learning is Transforming Video Game Development / gamedotro.com / - Текст: электронный URL: https://www.gamedotro.com/how-machine-learning-is-transforming-video-game-development/ (дата обращения: 15.03.2024)

8 Hanski Jari An Evaluation of the Unity Machine Learning Agents Toolkit in Dense and Sparse Reward Video Game Environments/ Hanski Jari, Biçak Kaan Baris. - urn:nbn:se:uu:diva-444982 - Текст: электронный - URL: https://www.diva-

portal.org/smash/record.jsf?pid=diva2%3A1563588&dswid=6302 (дата обращения: 16.03.2024).

9 Deep Reinforcement Learning to train agents in a multiplayer First Person Shooter/ Daniele Piergigli, Laura Anna Ripamonti, Dario Maggiorini, Davide Gadia. - https://doi.org/10.1109/CIG.2019.8848061- Текст: электронный // 2019 IEEE Conference on Games - URL: https://ieeexplore.ieee.org/abstract/document/8848061 (дата обращения: 16.03.2024).

10 The Future of machine learning in video games / aporia.com / - Текст: электронный URL: https://www.aporia.com/blog/the-future-of-machine-learning-in-video-games/ (дата обращения: 16.03.2024)

11 Aviv Elor Deep Reinforcement Learning in Immersive Virtual Reality Exergame for Agent Movement Guidance/ Aviv Elor, Sri Kurniawan. - https://doi.org/10.1109/SeGAH49190.2020.9201901 - Текст: электронный // IEEE 8th International Conference on Serious Games and Applications for Health - URL: https://ieeexplore.ieee.org/abstract/document/9201901 (дата обращения: 17.03.2024).

12 Download Unity / unity.com / - Текст: электронный URL: https://unity.com/download (дата обращения: 17.03.2024)

13 Machine Learning In Unity 3D / analyticsindiamag.com / - Текст: электронный URL: https://analyticsindiamag.com/everything-you-need-to-know-about-machine-learning-in-unity-3d/ (дата обращения: 17.03.2024)

14 Marius Matulis A robot arm digital twin utilising reinforcement learning/ Marius Matulis, Carlo Harvey. - https://doi.org/10.1016/j.cag.2021.01.011 - Текст: электронный // Computers & Graphics 106-114 - URL: https://www.sciencedirect.com/science/article/abs/pii/S009784932100011X (дата обращения: 17.03.2024).

15 TensorFlow / tensorflow.org / - Текст: электронный URL: https://www.tensorflow.org/?hl=ru (дата обращения: 17.03.2024)

16 Training an Agent for Third-person Shooter Game Using Unity ML-Agents/ Jun LAI, Chen Xiliang, Xue-zhen ZHANG. - http://dx.doi.org/10.12783/dtcse/icaic2019/29442- Текст: электронный // DEStech Transactions on Computer Science and Engineering - URL: https://www.researchgate.net/publication/333256414_Training_an_Agent_for_Third-person_Shooter_Game_Using_Unity_ML-Agents (дата обращения: 17.03.2024).
17 Getting started with ml agents in unity / hub.packtpub.com / - Текст: электронный URL: https://hub.packtpub.com/getting-started-with-ml-agents-in-unity-tutorial/ (дата обращения: 17.03.2024)
18 Andersson Pontus Future-proofing Video Game Agents with Reinforced Learning and Unity ML-Agents / Andersson Pontus. - https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1605238 - Текст: электронный // URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1605238&dswid=8168 (дата обращения: 17.03.2024).
19 An introduction to machine learning with Unity ML-Agents / dev.to / - Текст: электронный URL: https://dev.to/joooyz/an-introduction-to-machine-learning-with-unity-ml-agents-3an5 (дата обращения: 17.03.2024)
20 Mariana Werneck Generating Procedural Dungeons Using Machine Learning Methods/ Mariana Werneck, Esteban W. G. Clua. - https://doi.org/10.1109/SBGames51465.2020.00022 - Текст: электронный // 19th Brazilian Symposium on Computer Games and Digital Entertainment - URL: https://ieeexplore.ieee.org/abstract/document/9291584 (дата обращения: 18.03.2024).
21 Unity ML Agents / gamedev.net / - Текст: электронный URL: https://www.gamedev.net/articles/programming/engines-and-middleware/unity-ml-agents-r5114/ (дата обращения: 18.03.2024)
22 Baby Nirmal Implementing artificial intelligence agent within connect 4 using unity3d and machine learning concepts/ Baby Nirmal, Goswami, Bhargavi. -

ISSN: 2277-3878 - Текст: электронный // International Journal of Recent Technology and Engineering, 193-200. - URL: https://eprints.qut.edu.au/199647/ (дата обращения: 18.03.2024).

23 CUDA Toolkit / developer.nvidia.com / - Текст: электронный URL: https://developer.nvidia.com/cuda-toolkit (дата обращения: 18.03.2024)

24 Installation ML-Agents Toolkit / unity-technologies.github / - Текст: электронный URL: https://unity-technologies.github.io/ml-agents/Installation/ (дата обращения: 18.03.2024)

25 Unity ML-Agents Toolkit / unity-technologies.github / - Текст: электронный URL: https://unity-technologies.github.io/ml-agents/ (дата обращения: 19.03.2024)

26 Машинное обучение агентов в Unity / habr.com / - Текст: электронный URL: https://habr.com/ru/articles/454612/ (дата обращения: 19.03.2024)

27 Introduction on ML-Agents in Unity/ blog.yudiz.com / - Текст: электронный URL: https://blog.yudiz.com/dipping-your-toes-into-unity-ml-agents/ (дата обращения: 19.03.2024)

28 ML Agents/ docs.unity3d.com / - Текст: электронный URL: https://docs.unity3d.com/Manual/com.unity.ml-agents.html (дата обращения: 19.03.2024)

29 SARSA Reinforcement Learning / geeksforgeeks.org / - Текст: электронный URL: https://www.geeksforgeeks.org/sarsa-reinforcement-learning/ (дата обращения: 05.04.2024)

30 SARSA Algorithm in Python / annytab.com / - Текст: электронный URL: https://www.annytab.com/sarsa-algorithm-in-python/ (дата обращения: 05.04.2024)

31 SARSA Reinforcement Learning Algorithm / builtin.com / - Текст: электронный URL: https://builtin.com/machine-learning/sarsa (дата обращения: 05.04.2024)

32 SARSA in Reinforcement Learning / analyticsindiamag.com / - Текст: электронный URL: https://analyticsindiamag.com/a-complete-intution-on-sarsa-algorithm/ (дата обращения: 05.04.2024)
33 Proximal Policy Optimization / huggingface.co / - Текст: электронный URL: https://huggingface.co/blog/deep-rl-ppo (дата обращения: 15.04.2024)
34 Proximal Policy Optimization / spinningup.openai.com / - Текст: электронный URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html (дата обращения: 15.04.2024)
35 Proximal Policy Optimization / toloka.ai / - Текст: электронный URL: https://toloka.ai/blog/proximal-policy-optimization/ (дата обращения: 15.04.2024)
36 Soft Actor-Critic / spinningup.openai.com / - Текст: электронный URL: https://spinningup.openai.com/en/latest/algorithms/sac.html#id2 (дата обращения: 25.04.2024)
37 Soft Actor-Critic (SAC) Agents / mathworks.com / - Текст: электронный URL: https://www.mathworks.com/help/reinforcement-learning/ug/sac-agents.html (дата обращения: 25.04.2024)
38 Soft Actor-Critic: Re-Implementation and Experiments/ dosssman.github.io / - Текст: электронный URL: https://dosssman.github.io/posts/2022-04-13-sac/ (дата обращения: 25.04.2024)
39 Unity ML-Agents - An Introduction / xaviergeerinck.com / - Текст: электронный URL: https://xaviergeerinck.com/2020/05/27/unity-ml-agents---an-introduction/ (дата обращения: 25.04.2024)
40 TensorBoard / www.tensorflow.org / - Текст: электронный URL: https://www.tensorflow.org/tensorboard?hl=ru (дата обращения: 22.03.2024)
41 ML-Agents plays DodgeBall / blog.unity.com / - Текст: электронный URL: https://blog.unity.com/engine-platform/ml-agents-plays-dodgeball/ (дата обращения: 27.04.2024)

42 GAIL vs Behavioral Cloning / forum.unity.com / - Текст: электронный URL: https://forum.unity.com/threads/gail-vs-behavioral-cloning-whats-the-difference.944463/ (дата обращения: 27.04.2024)
43 Unity-Technologies / github.com/Unity-Technologies / - Текст: электронный URL: https://github.com/Unity-Technologies/ml-agents-dodgeball-env/ (дата обращения: 27.04.2024)
44 ML-Agents plays DodgeBall / blog.unity.com / - Текст: электронный URL: https://blog.unity.com/engine-platform/ml-agents-plays-dodgeball/ (дата обращения: 27.04.2024)
45 Enemy AI in Unity Games / pavcreations.com / - Текст: электронный URL: https://pavcreations.com/enemy-ai-in-unity-games-with-ml-agents-toolkit/2/ (дата обращения: 27.04.2024)