

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

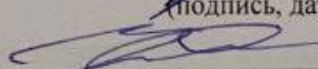
на тему

**ИССЛЕДОВАНИЕ СТРАТЕГИЙ В
КОМПЬЮТЕРНЫХ ИГРАХ С
ПРИМЕНЕНИЕМ МЕТОДОВ
ИСКУССТВЕННОГО ИНТЕЛЛЕКТА**

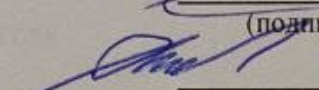
Студент

 31.05.24 Щегольков Михаил Валерьевич
(подпись, дата) (ФИО полностью)

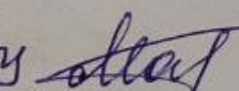
Руководитель

 01.06.24 Зинкин С.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

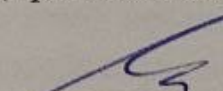
 01.06.24 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

зав. каф. Программирования ПензГУ 
(должность, место работы) (подпись, дата) 05.06.24 Мартышкин А.И.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой

 (подпись) Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от

06.06.2024 № 6)

Секретарь ГЭК

 (подпись) Гурин Е.И.
(фамилия, инициалы)

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

Факультет

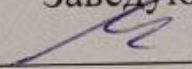
вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

 М.А. Митрохин

«13» сентября 2022 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Щегольков Михаил Валерьевич гр. 22ВВИм2 факультета вычислительной техники

2. Тема работы «Исследование стратегий в компьютерных играх с применением методов искусственного интеллекта»

Тема утверждена приказом ПГУ № 765/о от "13" сентября 2022 г.

3. Руководитель магистерской диссертации д.т.н., профессор Зинкин С.А.

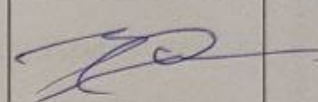
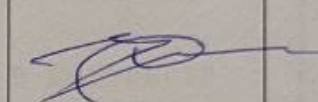
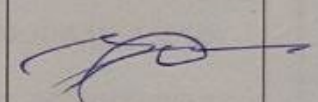
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать игру с внедрением в неё игрового искусственного интеллекта на основе нейронной сети, обученного на алгоритмах глубокого обучения. Исследовать особенности обучения агента и описать рекомендации для этого обучения.

5. Перечень подлежащих разработке вопросов:

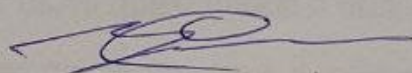
1. Анализ предметной области игрового искусственного интеллекта и возможностей его внедрения при разработке игр.
2. Разработка игры с внедрением в неё агента.
3. Исследование и анализ обучения агента.
4. Отладка игры и агента.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области игрового искусственного интеллекта	10%	13.09.2022 – 14.11.2022	
2	Анализ предметной области игр	10%	15.11.2022 – 16.01.2023	
3	Проектирование структуры разрабатываемой системы	15%	17.01.2023 – 20.04.2023	
4	Разработка всех компонентов игры	25%	21.04.2023 – 23.09.2023	
5	Обучение агента и исследование полученных результатов	20%	24.09.2023 – 26.01.2024	
6	Отладка разработанной системы	10%	27.01.2024 – 29.03.2024	
7	Оформление выпускной работы	10%	29.03.2024 – 31.05.2024	

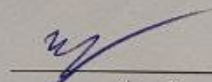
Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации


(подпись, дата)

Зинкин С.А.

Задание к исполнению принял студент


(подпись, дата)

Щегольков М.В.

Работу к защите допустить
Декан факультета

 03.06.2024
(подпись, дата)

Фионова Л.Р.

РЕФЕРАТ

Магистерская работа содержит 84 листа, 35 рисунков, 2 таблицы, 8 листингов, 32 источника, 2 приложения

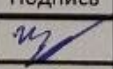
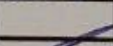
ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, ИГРОВОЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, НЕЙРОННЫЕ СЕТИ, ИГРОВАЯ СРЕДА, ПОВЕДЕНИЕ АГЕНТОВ

Цель работы – проектирование и создание игры с внедрённым в неё ИИИ. Анализ обучения агента на основе созданной игровой среды, формирование рекомендации для непосредственной реализации искусственного интеллекта в компьютерных играх.

Объектом разработки является искусственный интеллект в контексте игровой среды, предметом – глубокое обучение с подкреплением, которое будет реализовываться в рамках созданной игры.

В результате проведенной работы разработаны и протестированы все компоненты системы, включая игровой интерфейс и механики, обученные модели и сформированные рекомендации.

Результаты данной работы могут быть применены в области разработки игр, а также для дальнейших научных исследований в области обучения с подкреплением.

					ПГУ 1.090401.47.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата	<div>Исследование стратегий в компьютерных играх с применением методов искусственного интеллекта</div> <div>Пояснительная записка</div>		
Разраб.		Щегольков М.В.		31.05.24			
Провер.		Зинкин С.А.		01.06.24			
Н. Контр		Бычков А.С.		01.06.24			
Утверд.		Митрохин М.А.		03.06.24			
					Лит.	Лист	Листов
						2	84
					ФВТ, каф. ВТ гр. 22ВВИм2		

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Анализ предметной области	7
1.1 Общая характеристика игрового искусственного интеллекта	7
1.2 Методы реализации игрового искусственного интеллекта	8
1.3 Обучение с подкреплением. Обучение без знания модели	12
1.4 Общая характеристика нейронных сетей. Нейронные сети в обучении с подкреплением	17
1.5 Выбор средств для создания окружения	20
1.6 Выбор набора инструментов для написания внутренней логики и обучения агента	22
1.7 Выводы	24
2 Разработка приложения	25
2.1 Проектирование и создание окружения	25
2.2 Подготовка агента к обучению	31
2.3 Описание алгоритмов обучения агента	40
2.4 Анализ обучения агента	44
2.5 Описание приложения	56
2.6 Тестирование	58
2.7 Выводы	60
ЗАКЛЮЧЕНИЕ	61
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	63
ПРИЛОЖЕНИЕ А Листинг программы	66
ПРИЛОЖЕНИЕ Б Презентация	76

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

ИИ – Искусственный интеллект

ИИИ – Игровой искусственный интеллект

AR (Augmented reality) – Дополненная реальность

IDE (Integrated development environment) – Интегрированная среда разработки

ML (Machine Learning) – Машинное обучение

NPC (Non-player character) – Неигровой персонаж

PPO (Proximal policy optimization)

ReLU (Rectified linear unit) – Линейный выпрямитель[

RL (Reinforcement learning) – Обучение с подкреплением

SAC (Soft Actor-Critic)

VR (Virtual reality) – Виртуальная реальность

ВВЕДЕНИЕ

На сегодняшний день искусственный интеллект используется во многих областях и затрагивает множество сфер человеческого общества, начиная от развлечений и проходя через медицину, образование, экономику и другие сферы. Если изначально ИИ отдалённо затрагивал эти области, то уже сейчас он всё глубже проникает в нашу реальность и становится одной из наиболее актуальных проблем. Благодаря развитию области искусственного интеллекта уже сейчас возможно создание уникальных систем, которые направлены на решение задач, не всегда посильных для человек.

Область компьютерных игр является одной из возможных сфер реализации искусственного интеллекта. Изначально игры использовались только как программы для развлечения и отдыха, но это уже давно не так, игры могут использоваться и в образовательных целях, и, с активным развитием ИИ, всё чаще используются для моделирования реальных процессов с последующим переносом обученных агентов. В области игр искусственный интеллект может проявляться в различных вариантах, как для улучшения графики и определения поведения неигровых персонажей, так и для затронутого выше моделирования реальных процессов, которые легче и эффективнее реализовать сначала в игровых ситуациях. Сфера компьютерных игр является наиболее доступной и восприимчивой к новым технологическим решениям в области обучения с подкреплением, так как позволяет удешевить, ускорить и полнее контролировать обучение модели, проверяя тем самым многие задумки до материального воплощения.

В данной работе будет рассматриваться обучение с подкреплением для создания игрового искусственного интеллекта у агента, помещённого в специально разработанную среду и выполняющего определённую задачу. Описанная выше модель позволит улучшить игровой опыт игроков, благодаря внедрению нейронной сети, а также может быть ориентирована на последующее внедрение в сферы реального мира.

В магистерской диссертации объектом исследования выступает игровой искусственный интеллект. Предметом исследования является глубокое обучение с подкреплением и компьютерные игры.

Целью магистерской диссертации является создание игры с внедрённым в неё ИИИИ, рассмотрение и анализ различных алгоритмов ИИ и архитектур нейронной сети, формирование рекомендации для непосредственной реализации искусственного интеллекта в компьютерных играх.

1 Анализ предметной области

Создание любой научной работы или программного продукта необходимо начинать с теоретического анализа, рассмотрения существующего инструментария, разделения существующей задачи на последовательные подзадачи, следующие друг за другом и ведущие к получению наиболее эффективного решения всей системы, путём проработки каждого шага, опираясь на результаты, полученные на предыдущем шаге. Первым таким шагом является исследование общих характеристик ИИИ и методов его реализации.

1.1 Общая характеристика игрового искусственного интеллекта

Искусственный интеллект – программа-система, которая имитирует процесс мышления человека. Имитация данного процесса может выражаться в решении определённых задач, принятии решений или выполнении творческих функций [1].

Игровой искусственный интеллект – это вид искусственного интеллекта, который имитирует наличие интеллекта у NPC в игровых ситуациях. Не всегда игровой ИИ реализуется с использованием методов академического ИИ, часто он состоит из множества эвристик и эмпирических правил, но ИИИ по-прежнему может рассматриваться как подотрасль ИИ. Целью использования ИИИ является искусственная замена человеческого мышления и имитация человекоподобного поведения в игре, без непосредственного участия людей [2].

Основной целью развития игрового ИИ выступает достижение такого уровня, чтобы игрок не был в состоянии отличить соперника-компьютера от соперника-человека.

Игровой ИИ может проявляться как:

- то, что делает окружение реалистичнее, а графику красивее,
- то, что анализирует привычки и поведение игроков,
- то, что наделяет мышлением или его подобием неигровых персонажей (NPC) или ботов [3].

В нашем исследовании будет затрагиваться 3 пункт. Игровой искусственный интеллект может реализовываться с помощью большого

количество разнообразных методов разной сложности и функциональности, рассмотрим наиболее часто встречающиеся.

1.2 Методы реализации игрового искусственного интеллекта

Начнём рассмотрение с наиболее простого метода – rule-based ИИ или жестко заданные условные конструкции. Данный подход представляет собой простой набор правил, который позволяет NPC реагировать на окружающее состояние и незамедлительно принимать решение как действовать. С помощью данного подхода можно создать простое поведение [4].

Деревья решений – система, представленная в форме дерева для нахождения решений. Узлы данной системы могут быть представлены либо узлами решений, которые предлагают какой-либо условный выбор, либо конечными узлами, представляющими конкретные действия и являющимися решениями дерева. Тогда задачей узлов дерева является перевод игровой системы в конечное состояние. Деревья решений похожи на простые условные конструкции, рассмотренные выше, но они позволяют автоматизировать реализацию этих условных конструкций. Пример дерева решений показан на рисунке 1 [5].



Рисунок 1 – Пример дерева решений

Другой часто используемый метод для принятия решений – реакция на события. Приведённые выше методы предполагают постоянный опрос для выполнения действия в соответствии с условием, реакция на событие

представляет собой простую реактивную систему, которая делится на событие и реакцию на это событие.

Описанные техники используются для простых принятий решений исходя из заданных условий, они просты в реализации и могут использоваться только для задания относительно простого поведения NPC. Рассмотрим более сложные техники, которые позволят реализовать более «интеллектуальное» поведение.

Конечные автоматы (finite state machine). Этот подход позволяет игровым агентам менять в течение игры различные состояния. Для данной системы задаётся два множества: конечного количества состояний и правил перехода из одного состояния в другое. Тогда каждое состояние – это сценарий поведения, который будет или не будет выполнять переход к другому состоянию. Абстрактное графическое представление конечного автомата представлено на рисунке 2 [6].

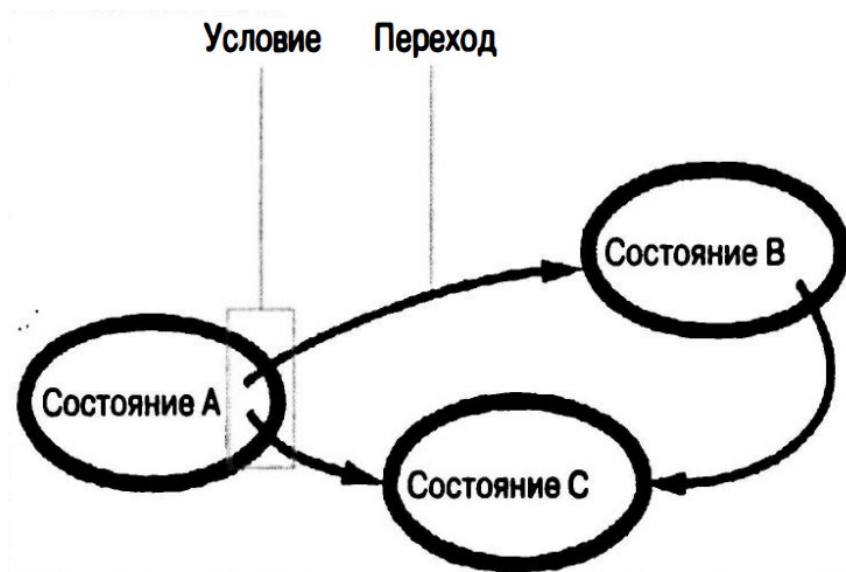


Рисунок 2 – Конечный автомат

Улучшением конечных автоматов являются иерархические конечные автоматы. В данном варианте вся система может иметь иерархию или слои, то есть каждое состояние может представлять собой отдельный конечный автомат. Данная особенность позволяет группировать состояние, что избавляет от добавления большого числа лишних переходов и делает систему более масштабируемой и гибкой [6].

Принятие решений через переходы в описанных выше конечных автоматах непосредственно связано с текущим состоянием, если необходимо принятие решений в независимости от состояния могут использоваться деревья поведений. В такой системе решения находятся отдельно от состояний. Данный метод похож на дерево решений, но и имеет серьёзные отличия:

- результатом узла является одно из трёх возможных значений: «успешно», «неуспешно» или «выполняется»,
- вместо узлов решений с двумя вариантами в системе присутствуют узлы-«декораторы» с одним дочерним узлом, который выполняется в случае «успешного» вывода «декоратора»,
- для обозначения происходящих действий в узлах действий возвращается «выполняется».

Деревья поведений – мощный и довольно сложный инструмент с множеством различных способов составления и реализации дерева [7].

Другая техника – система на основе полезности. Она имеет меньшую мощность, по сравнению с деревьями поведения, но позволяет иметь множество различных действий, выбор которых основан на полезности каждого из них в конкретный момент времени.

Также важно затронуть планирование в игровом ИИ. Самый простой планировщик может быть реализован как перебор действий, однако данный подход очень неэффективный. Вместо него можно использовать, например «обратный обход», когда вместо того, чтобы проверять все действия и их результаты, можно начать рассмотрение с конечных желательных результатов и искать возможный к ним путь.

Другой метод – поиск по первому наилучшему совпадению. Данный алгоритм разворачивает цепочку действий для наиболее подходящих узлов, вместо рассмотрения всех. Его вариантами могут выступать поиск A^* или метод Монте-Карло, который многократно повторяя процесс расчёта результата, меняет веса вариантов действий и даёт оценку следующего шага [8].

Также можно выделить Goal-Oriented Action Planning (GOAP) или планирование действий на основе целей. Данный метод начинается с цели и подбирает шаги, приводящие к ней [7].

Выше были рассмотрены основные методы реализации игрового искусственного интеллекта, некоторые из них используются для имитации совсем простого поведения, а некоторые имеют довольно сложную структуру, которая обеспечивает и более сложное поведение. Для более «интеллектуального» поведения необходимо обучение в процессе игры, которое реализуется с помощью адаптивного ИИИ.

Реализации игрового ИИ детерминированными правилами приведёт к тому, что игрок через время запомнит и выучит заложенное поведение, и найдёт оптимальную стратегию. Для игр, где необходимо избежать данной ситуации и оставлять игровое поведение непредсказуемым необходимо, чтобы игровые агенты адаптировались к игре человека, а для этого необходимо предсказывать его следующее действие. Предсказания могут быть основаны на оценке предыдущих важных решений противника ИИ в данной сессии и на основе ряда предыдущих игр, обеспечивая тем самым приспособление к стилю игры человека.

Как наиболее простые методы для реализации такого подхода можно использовать простую вероятность того или иного поведения, которая будет зависеть от решений игрока в предыдущих сессиях или использовать наивный байесовский классификатор. Информацию о противнике и всю окружающую информацию можно подавать на вход данного классификатора, который будет классифицировать сложившуюся обстановку и подбирать наиболее подходящее поведение [2].

Можно усложнить данную систему, используя изменение набора значений, который обеспечивает выбор стратегии, то есть можно собирать всю окружающую информацию и в зависимости от неё изменять веса для тех или иных решений. Сам выбор будет случайным, но зависящим от весов. Используя данный метод для прогноза поведения игрока – мы получаем марковскую модель, в основе которой лежит марковское свойство – будущее состояние зависит только от

текущего, а не от предыдущих. Использование цепи Маркова для предсказания является аппроксимацией и добавление дополнительных факторов в принятие решения человеком отрицательно скажется на полученном прогнозе [9].

Если для прогноза необходимо анализировать последовательность событий, то можно воспользоваться алгоритмом N-грамм. Последовательность событий и будет являться N-граммой, где N – количество событий, которые запоминаются. В методе N-грамм хранится информация о вероятностях действий, прогноз строится как учёт всех комбинаций вариантов предыдущих ходов. Малое N может не дать ожидаемых результатов, то есть качество прогноза будет небольшим, а большое N значительно увеличивает количество комбинаций, что отражается на объёме памяти и их анализе [6].

Здесь были рассмотрены достаточно простые и популярные методы для реализации адаптивного игрового искусственного интеллекта с присущими им особенностями. Они уже показывают способность приспосабливаться к стилю игры оппонента или к окружающей обстановке, но их адаптация находится в заранее заложенных разработчиком рамках.

1.3 Обучение с подкреплением. Обучение без знания модели

Описанных в предыдущих пунктах методов достаточно для реализации ИИИ в большинстве проектов, но существуют более сложные подходы, использование которых позволяет добиться более эффективных результатов, которые могут использоваться, в том числе, вне рамок игр.

Обучение с подкреплением – метод, в котором используется корректировка стратегии по принципу проб и ошибок через обратную связь с данными, поступающими из окружающего мира, то есть ставятся задачи, в которых агент должен выполнять последовательность действий в определённом окружении, с получением прямой обратной связи о результатах своих действий. Использование обучения с подкреплением позволяет реализовывать адаптивное поведение при этом не опираясь на сценарии или правила, установленные разработчиком. При непосредственной реализации алгоритмов обучения с подкреплением для

усвоения наиболее оптимальной стратегии или действия используется вознаграждение от среды [10].

Взаимодействие агента и среды является базовой концепцией для методов обучения с подкреплением, поэтому необходимо рассмотреть это взаимодействие поподробнее. На шаге t агент получает состояние среды S которое он непосредственно наблюдает, и награду за предыдущее действие R . Состояние среды определяет следующее действие агента A , при этом действия агента направлены на максимизацию получаемой награды. Действие агента меняет среду (среда может меняться и под другими факторами), состояние которой вместе с наградой приходит на шаге $t+1$, формируя таким образом всё поведение агента. Схема взаимодействия между агентом и средой представлена на рисунке 3 [11].

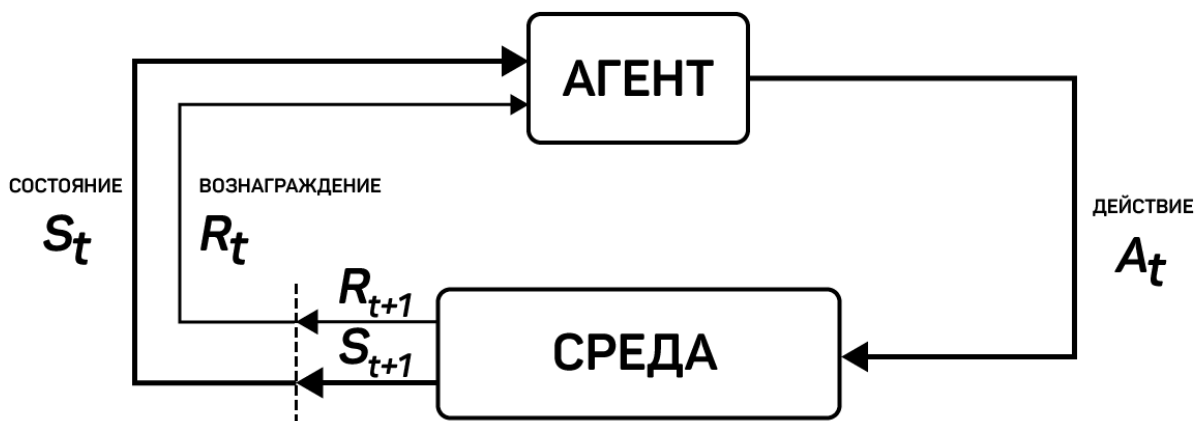


Рисунок 3 – Взаимодействие между агентом и окружающей средой

В обучении с подкреплением можно выделить 2 основные группы алгоритмов – обучение без знания модели (Model-Free RL), которые являются простыми и универсальными, так как не требуют построения модели окружения, и обучение со знанием модели (Model-Based RL), для которых необходимо явно прописывать модель среды. Мы рассмотрим более подробно Model-Free RL, в этой группе выделяют 2 основных подхода к обучению: Q-Learning и Policy Optimization. Пример классификации представлен на рисунке 4 [12].

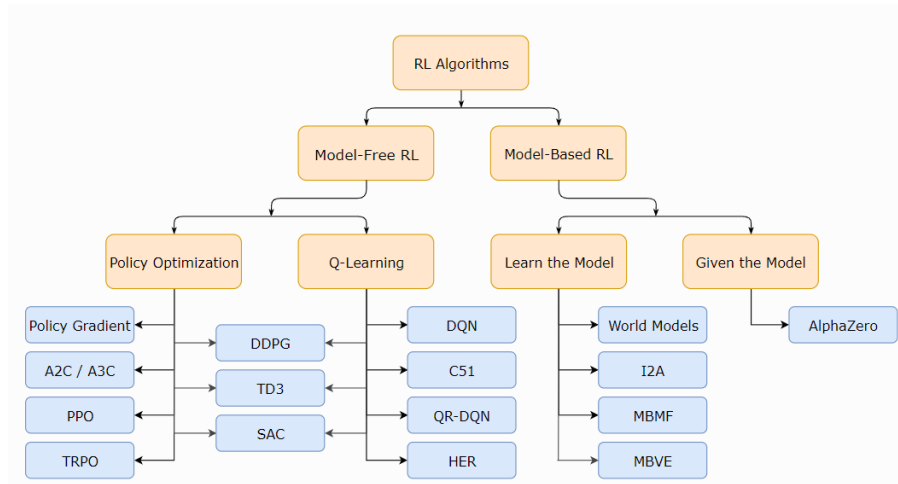


Рисунок 4 – Классификация алгоритмов обучения с подкреплением

Задачу обучения через взаимодействие со средой можно свести к некоторой функции, которая бы принимала на вход состояние среды и возвращала бы вероятность действия агента, такая функция называется policy function (функция политики). Policy Gradient используется именно такую функцию. Этот алгоритм представляет собой градиентный спуск, только здесь исход игры используется вместо правильного ответа, а действия агента будут попадать под изменения весов. Данный алгоритм можно использовать в рамках глубокого обучения, то есть с использованием нейронной сети, тогда функция потерь (L_{PG}) будет иметь следующий вид

$$L_{PG} = -\sum_i r_i \ln p(c = a_i | s_i), \quad (1)$$

где r_i – награда на шаге i ;

p – функция политики;

a_i – действие на шаге i ;

s_i – состояние среды на шаге i [13].

На практике выработаны некоторые приёмы, которые обеспечивают лучшее схождение для модели. Один из таких приёмов – discounted rewards, данный приём предполагает при вычислении награды (R_s) использование коэффициента дисконтирования (γ)

$$R_s = \sum_{i=0 \dots n-s} \gamma^i r_{s+i}, \quad (2)$$

где γ – коэффициент дисконтирования;

r – награда.

Коэффициент дисконтирования используется из ожидания того, что наиболее близкая награда является более важной для обучения, так как нейронная сеть может повлиять на неё сразу.

Следующим важным приёмом можно выделить Baseline, он предполагает введения значения (b), вычитаемого на каждом шаге из награды (R_s)

$$R_s = \sum_{i=0 \dots n-s} \gamma^i r_{s+i} - b, \quad (3)$$

где γ – коэффициент дисконтирования;

r – награда;

b – baseline.

Значение baseline базовым значение награды, которое, начиная с определённого шага, уже достигнуто. Использование данного приёма позволяет более правильно учитывать награду.

Но чётко определённый baseline не всегда даёт ожидаемые результаты, более интересным выглядит вариант использования динамического baseline, определяемого для каждого состояния. Это идея развивается в алгоритме Actor-Critic, вводится функция значения, которая возвращает ожидаемую награду, получая на вход состояние среды. Данная функция также может быть обучена нейронной сетью, более того, очень часто она объединена в одну нейросеть вместе с policy function образуя как бы 2 стороны – того, кто действует (Actor) и того, кто критикует эти действия (Critic). Сначала обучается Critic, часть, которая предсказывает ожидаемую награду, а после Actor обучается на основе оценки Критика. Данный алгоритм показывает интересную систему, где каждая часть подстраивается под другую, при этом обучаясь и продолжая максимизировать желаемые значения [12].

Также можно выделить алгоритмы PPO, эти алгоритмы также основаны на функции политики, но их главной особенностью является введение проксимальных ограничений для контроля изменений, для этого вводится коэффициент, ограничивающий отношения между старой и новой стратегиями, что обеспечивает более стабильно обучение.

Другая группа алгоритмов основана на оценки ожидаемой суммарной награды в состоянии s , если при этом агент примет действие a – Q-функция. Зная при этом оптимальную Q-функцию мы бы просто могли выбирать действие приносящее максимальную награду, получить Q-функцию оптимально стратегии в большинстве случаев мы не можем, но используя уравнение Беллмана, мы можем итеративно приближать Q-функцию [14].

Параметризация данной функции весами нейронной сети приведёт к тому, что она должна будет предсказывать усреднение самой себя на шаг вперёд, чтобы получать немного более оптимальные значения. Так как наличие Q-функции оптимальной стратегии позволяет использовать её же на последующий состояниях, усредняя результат мы будем получать оптимальный выход для данного шага. Важной особенностью данного алгоритма выступает возможность использовать данные других обученных агентов [15].

Примером существующих продуктов, которые реализовывают обучение с подкреплением в видеоиграх, можно привести DeepMind DQN, которая получает только состояние, вознаграждение и возможные действия, училась играть в игры для Atari. Данный алгоритм превзошёл другие в 94% игр, а на многих (в 59% игр) оказался выше уровня человека. Другим примером может служить AlphaStar – программа, играющая в StarCraft 2, которая стала гроссмейстером игры, обыграв 99,8% игроков [16].

Если в работу алгоритма обучения с подкреплением добавить искусственную нейронную сеть, то мы подойдём к методу глубокого обучения с подкреплением. Комбинация глубокого обучения и обучения с подкреплением оказывается очень сильной для решения многих задач, особенно, когда отдельные эксперименты дорогие и посредством метода проб и ошибок можно улучшать

стратегию огромным количеством шагов обучения. Но тут возникают свои минусы, которые заключаются в требованиях к большому количеству данных и вычислительной мощности, при этом обученные модели зачастую направлены на одну специфическую задачу [17].

1.4 Общая характеристика нейронных сетей. Нейронные сети в обучении с подкреплением

Нейронные сети – математическая модель из области искусственного интеллекта, которая воплощает биологические нейронные сети мозга живых существ. Математические нейронные сети прошли большой эволюционный путь, имеют разнообразную архитектуру и сферы применения, но общими механизмами и идеями продолжают напоминать реальные нейронные сети.

Базовой структурой нейросети является нейрон, который представляет собой минимальный блок, принимающий входные сигналы, нейрон обрабатывает их и генерирует выходной сигнал. Общая структура искусственного нейрона представлена на рисунке 5 [18].

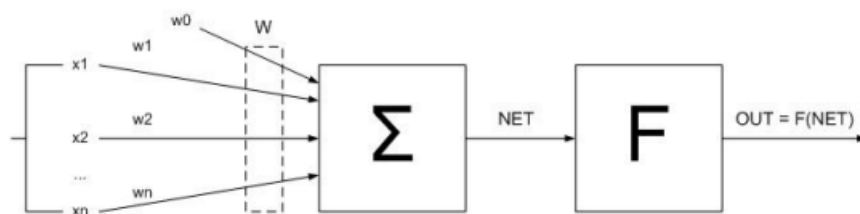


Рисунок 5 – Искусственный нейрон

Нейроны объединяются в слои, которых в большой нейросети может быть огромное количество. Все слои можно разделить на 3 группы:

- входной слой – слой, который занимается обработкой входных данных,
- скрытые слои, выполняющие всё основную обработку внутри сети,
- выходной слой обрабатывает результат работы сети в формат выходного сигнала, который ожидается увидеть.

В нейросети по одному входному и выходному слою, остальное место занимают скрытые слои. Так же важно выделить механизм обработки сигнала и генерации выходного значения, за это отвечает функция активации, которая просто говорит нужно ли передавать сигнал дальше. Существует множество

разных функция: линейная, ступенчатая, сигмоида и так далее, но самой часто применяемой в настоящее время является ReLU с различными модификациями, в самой просто вариации ReLU возвращает значение x , если оно положительно, в противном случае 0. График функции приведен на рисунке 6 [19].

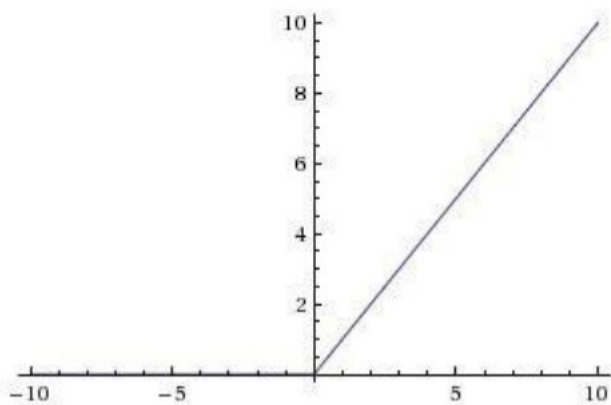


Рисунок 6 – График функции ReLU

Нейроны связаны между собой, эти связи могут иметь различный вид, но чаще всего на практике мы видим полносвязные системы, когда каждый нейрон одного слоя связан с каждым нейроном соседнего слоя. Связи между нейронами имеют веса, которые определяют важность сигнала от источника этой связи, именно эти значения изменяются при обучении. Определение количества слоёв, числа нейронов в слоях, функции активации и как связаны нейроны определяют архитектуру нейронной сети. Обобщённая схема простой нейронной сети представлена на рисунке 7 [20].

В машинном обучении можно выделить несколько способов обучения: обучение с учителем, обучение без учителя и обучение с подкреплением.

Обучение с учителем – обучение происходит на размеченном наборе данных, то есть для каждого объекта в обучающей выборке есть правильный ответ по целевому значению. В данном случае обучение сети происходит путём сравнения своих ответов с правильными результатами, в таком случае с целью минимизации ошибки происходит изменение весов.

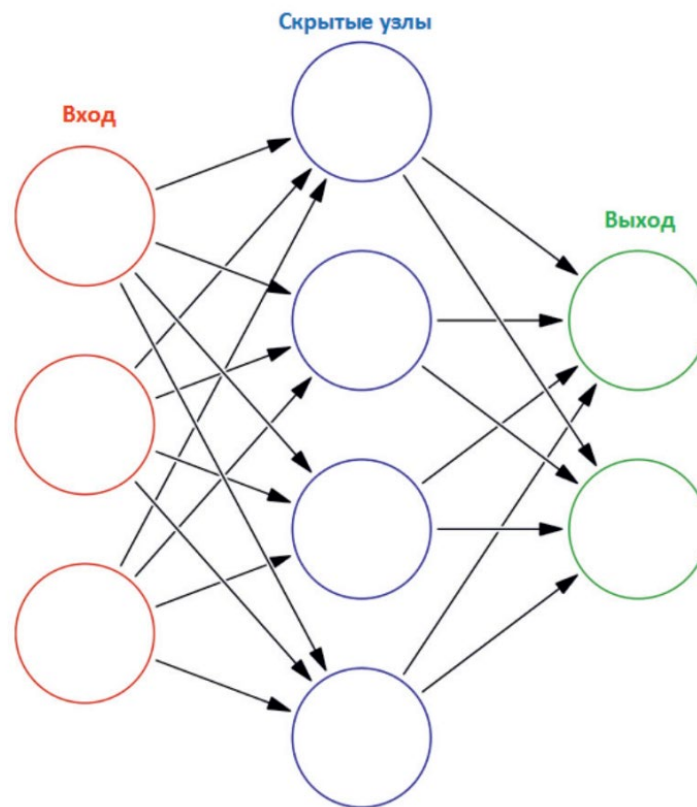


Рисунок 7 – Схема простой нейронной сети

Обучение без учителя. В этом виде обучение происходит без наличия правильных ответов, целью обучения в данном случае является выявление закономерностей между данными, примером задачи – кластеризация данных.

Обучение с подкреплением – в виде обучения у нас есть агент, который взаимодействует со средой, получая её состояния, агент совершает действия и получая ответ на эти действия в виде награды, изменяя при этом среду.

Как уже было написано выше обучение представляет из себя корректировку весов, для этого используется функция потерь, которая в общем виде оценивает разницу между предсказанным значением и реальным результатом, в случае с обучением с подкреплением функция будет иметь немного другой смысл, такая функция потерь описана выше и представлена в формуле 1.

В данном исследовании мы будем использовать глубокое обучение с подкреплением. Опишем входные и выходные сигналы данной сети. Входными сигналами, как уже было указано выше будут значения, описывающие состояние среды, но это описание может представляться в разных формах. Если наше

состояние представлено в виде картинку мы можешь передавать эту картинку, тогда для обучения будет использоваться свёрточная нейронная сеть. «Свёрточная нейронная сеть – специальная архитектура нейронной сети, в которой используются convolution layers и pooling (subsampling) layers. Данный тип нейронных сетей был предложен для распознавания образов, для чего эффективно и используется» [21]. Особенность данной нейросети в том, что используется свёртка, то есть каждая часть изображения умножается на ядро поэлементно, суммируя результат мы получаем выходное изображение. Для такой нейронной сети можно использовать Transfer learning – использовать уже обученную нейросеть, заморозив большую часть первых слоёв, а обучать только небольшую часть, что значительно ускорит обучение. Использование изображения как состояния среды ограничивает обучение, а также несмотря на Transfer learning утяжеляет его и делает более узконаправленным на ситуацию [22].

В рамках данного исследования будет использоваться набор входных значений, которые агент может отслеживать независимо от картинки, например, своё положение или расстояние до какого-либо объекта. Выходными значениями нейронной сети будет необходимость совершить то или иное действие из списка возможных действий.

1.5 Выбор средств для создания окружения

При изучении и разработки RL системы окружение является ключевым этапом, так как всё обучение будет происходить исходя из этого окружения и из того, как агент будет взаимодействовать с средой. Можно выделить несколько вариантов получения окружения.

Первым вариантом выступает использование готовых окружений, предоставляемых в рамках больших пакетов. Наиболее популярные из таких вариантов – OpenAI Gym – набор инструментов с открытым исходным кодом, который предоставляет интерфейс для глубокого и обычного обучения с подкреплением, данный инструмент представляет различные окружения – симуляторы роботов, текстовые игры, а также ряд игр Atari 2006. Примеры из окружения OpenAI Gym показаны на рисунке 8 [23].



Рисунок 8 – Примеры из окружения OpenAI Gym, где а – CartPole(задача из теории управления), b - LunarLander (задача управления в двумерном симуляторе)

Другое пакет – DeepMind Lab, которое построено на основе Quake 3 Arena, имитируя трёхмерный мир, в котором агент воспринимает всё от первого лица.

Данные пакеты предоставляют большой готовый функционал для использования различных сред и обучения агента в них, обеспечивая быструю разработку, но при этом очень сильно ограничивая в настройках и функциональности, зачастую такие среды имеют заданный порог для обучения и не позволяют подстроить окружение под возникающие требования.

Инструменты OpenAi Gym использовались для проведения небольшого числа исследований, но они упирались в ограничение, описанные выше.

Другой вариант – разработка собственной среды с нуля, без использования движков. В рамках данного подхода можно получить полный контроль над средой, изменяя и регулируя её под все требования в рамках задачи. Но главным минусом данного варианта является огромное количество затрачиваемых ресурсов на разработку, несопоставимую с результатами, так как современный игровые движки также предоставляют большой спектр возможностей в настройке и разработки окружения, имея при этом уже разработанные технологии, покрывающие многие потребности.

Третий вариант представляет собой нечто среднее между первым и вторым – это разработка собственной среды с использованием игровых движков. Данный вариант покрывает все потребности, так как позволяет в достаточной мере контролировать среду и обучение, а также предоставляет готовые инструменты

для создания графики, физики, полноценного дизайн-менеджмента, что позволят упростить и ускорить разработку. Игровых движков зарекомендовавших себе не так много, можно выделить Unity, Unreal Engine, Godot Engine.

Проанализировав плюсы и минусы имеющихся вариантов, Unity был выбран как оптимальный движок для исследования RL.

- Unity имеет мощные инструменты для создания игры, при этом позволяет делать этот относительно просто,

- данный движок имеет широкое сообщество разработчиков и полноценную документацию,

- для Unity существуют полноценные компоненты для реализации обучения с подкреплением,

- Unity предоставляет встроенный набор ассетов (готовых моделей, анимаций, материалов и т.д.), разработанных другими людьми, в том числе бесплатных,

- Unity имеет бесплатную версию для небольших проектов.

Для разработки среды будет использоваться Unity версии 2023.2.20f.

1.6 Выбор набора инструментов для написания внутренней логики и обучения агента

Следующим важным этапом после выбор инструментов создания окружения является выбор набора инструментов для написания внутренней логики и обучения агента. Набор инструментов в нашем случае ограничивается выбором среды.

Так как мы выбрали игровой движок Unity, то для написания кода будет использоваться основной язык, который используется в связке с Unity – C#. C# – мощный, гибкий и универсальный язык программирования, разработанный компанией Microsoft. В Unity используется 9 версия языка C# и компилятор Roslyn [24]. IDE в которой будет происходить написание кода – Microsoft Visual Studio, это мощная среда разработки, имеющая большой функционал для разработки в разных направлениях.

Unity имеет эффективный инструмент для обучения с подкреплением, который встраивается в движок, позволяет обучать агента прямо из редактора, а также предоставляет инструменты для настройки обучения и визуализации результатов. Unity Machine Learning Agents Toolkit (ML-Agents) – это проект с открытым исходным кодом, который предоставляет возможности играм быть средой для обучения игровых агентов. Данный проект предоставляет реализации основных алгоритмы обучения с подкреплением (построенные на базе PyTorch) для интеграции в 2D, 3D и VR /AR игры. ML-Agents предоставляет простой в использовании Python API для обучения агентов с использованием обучения с подкреплением, имитационного обучения, нейроэволюции или других методов [25].

Данная библиотека имеет стабильную релиз версию 21, выпущенную 9 октября 2023 года, однако в данной версии используются устаревшие библиотеки и она может ставится с конфликтами, поэтому был сделан выбор использовать develop версию с обновлением до последнего коммита a6bffbfb от 3 мая 2024 года.

Данный набор инструментов предоставляется для языка C#, но строится на базе Python, поэтому для его установки необходимы ряд компонентов. Для удобства работы с Python в рамках исследования используется Anaconda – дистрибутив языка Python, включающий прямо из коробки набор наиболее часто используемых библиотек для областей науки о данных и машинного обучения с разрешение зависимостей, а также ряд инструментов и средств, например, conda – их собственный менеджер зависимостей [26].

Для работы с ML-Agents используется python версии 3.10.12. Данная библиотека использует ряд зависимостей, опишем здесь основные, которые важны в рамках обучения с подкреплением:

- Numpy (1.23.5) – библиотека для работы с многомерными массивами и матрицами,
- Pillow (10.3.0) – инструмент для обработки изображений,
- Torch (PyTorch) (2.2.2) – фреймворк глубокого обучения, представляющий большой функционал для создания, обучения и развёртывания нейросетей,

– TensorBoard (2.16.2) – инструмент визуализации обучения нейронных сетей, дающий возможность мониторинга и отладки процесса обучения,

– Onnx (1.15.0) – библиотека для работы с форматом ONNX (Open Neural Network Exchange) – формат, для обмена моделями машинного обучения.

1.7 Выводы

В этом разделе было проведено полное исследование предметной области, в рамках которой будет разрабатываться среда с применением глубокого обучения с подкреплением, включая общие характеристики игрового искусственного интеллекта и методов его реализации, рассмотрено обучение с подкреплением и выбраны наиболее подходящие алгоритмы – алгоритмы без знания модели, которые позволяют обучать агента без прописывания среды, используя только опыт. Для обучения с помощью RL были выбраны нейронные сети, соответственно исследованы их общие характеристики и как они будут использоваться в рамках обучения с подкреплением. Были проанализированы возможные варианты использования игровых сред и решено создавать свою среду для полного контроля и установки своих характеристик, а также получение готового программного продукта, для этой цели будет использоваться Unity вместе с библиотекой ML-Agents, которая даёт возможность интегрировать обучение и внедрение нейронной сети в Unity. Основным языком разработки – C#.

2 Разработка приложения

Данная система будет состоять из двух частей: самой игры (среды) и обученного в данном окружении агента. Первым шагом будет выступать проектированием такой среды, в которой бы смог учиться агент, при этом среда должна носить элементы случайного характера, что бы агент не просто запоминал один удачный путь.

2.1 Проектирование и создание окружения

Важно создать такую игру, в которую мог бы играть и реальный человек, и обученный агент, с возможным вариантом их одновременного взаимодействия. Игра должна получиться как независимый продукт, который можно использовать вне этого исследования, в данной игре должна быть возможность развития, добавления новых игровых элементов, режимов и так далее, включая возможность добавления искусственного игрового интеллекта в эти новые обновления. Данная игра должна в какой-то мере отражать реальный мир, чтобы это как бы отражало действия агента, если бы он был перенесен из игровой среды в реальную.

Было решено взять переход через дорогу как основную концепцию игры. Подобные игры были популярны более 10 лет назад, но они были реализованы, в основном, в 2D графике с пошаговым перемещением игрока. Данная игра будет иметь 3D графику, с непрерывными изменениями состояния среды и возможностью непрерывных действия со стороны игрока/агента. Основная цель игры – перейти дорогу за персонажа, он начинает с одного края дороги и должен дойти до другого, сохранив жизнь. Пройти необходимо через несколько линий дорог, основным препятствие на которых являются машины, также персонаж неудачно заканчивает игру, если заходит на другие большие объекты.

Начнём с построения уровня: две стороны разделённые четырнадцатью линиями дороги, по которым перемещаются опасные объекты, между линиями дороги есть безопасная одна полоса, уровень на основе этого описания изображен на рисунке 9.

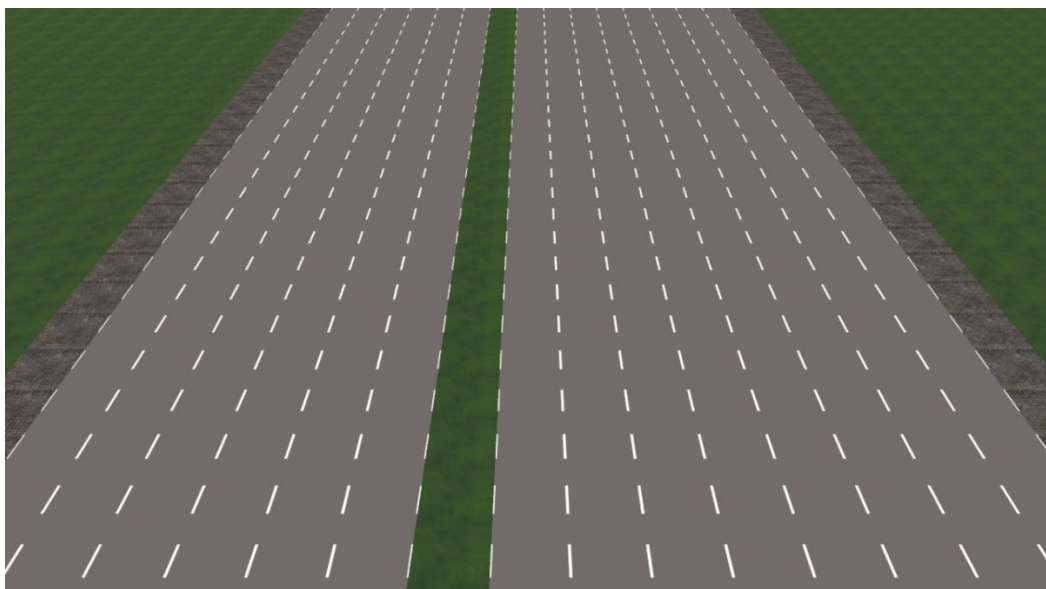


Рисунок 9 – Начальное оформление уровня

Далее рассмотрим префаб для персонажа, он изображен на рисунке 10. Префаб в Unity – это своего рода шаблон, который состоит из множества игровых объектов и может использоваться во многих местах игры. Персонаж имеет множество отдельных элементов для правильной анимации, скрипты для поведения, для обучения, а также скрипт для датчика лучей, чтобы «видеть» другие объекты. Эти скрипты будут подробнее рассмотрены ниже. У данного персонажа используется 2 анимации:

- ожидания, когда персонаж просто стоит без действий, она же изображена на рисунке 10,
- бега, когда персонаж перемещается.

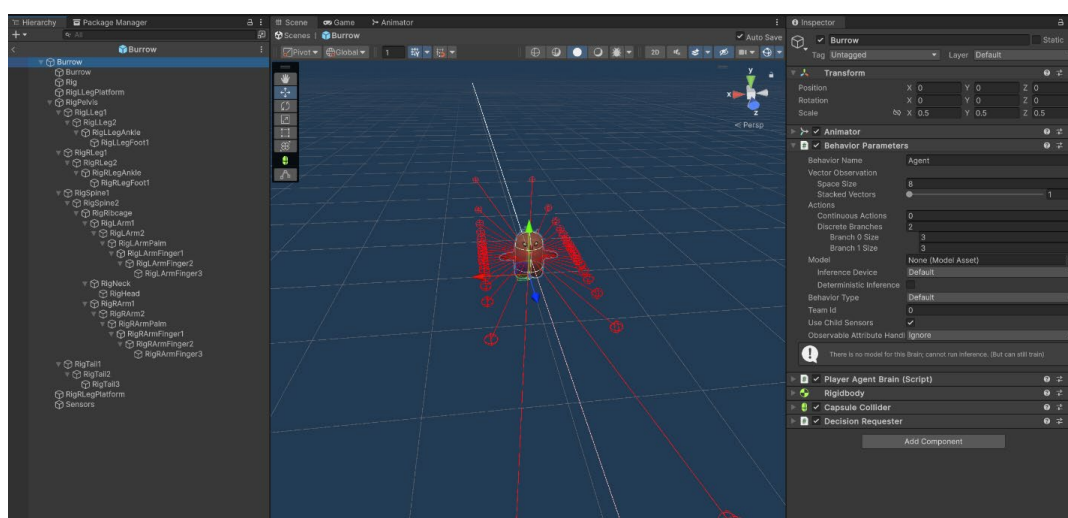


Рисунок 10 – Префаб персонажа

Далее для нашего приложения необходимы объекты, которые будут мешать персонажу пройти дорогу – машины. В игре используется 8 префабов машин, которые отличаются формой и различными деталями. Примеры префабов стандартной машины и автобуса приведены на рисунках 11 и 12 соответственно.

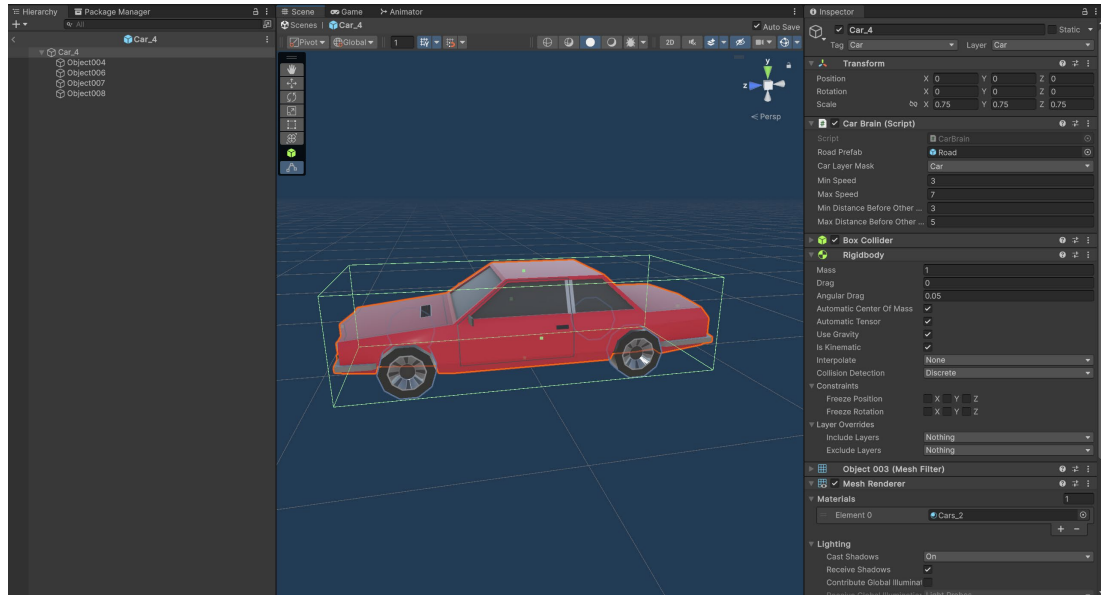


Рисунок 11 – Префаб с примером стандартной машины

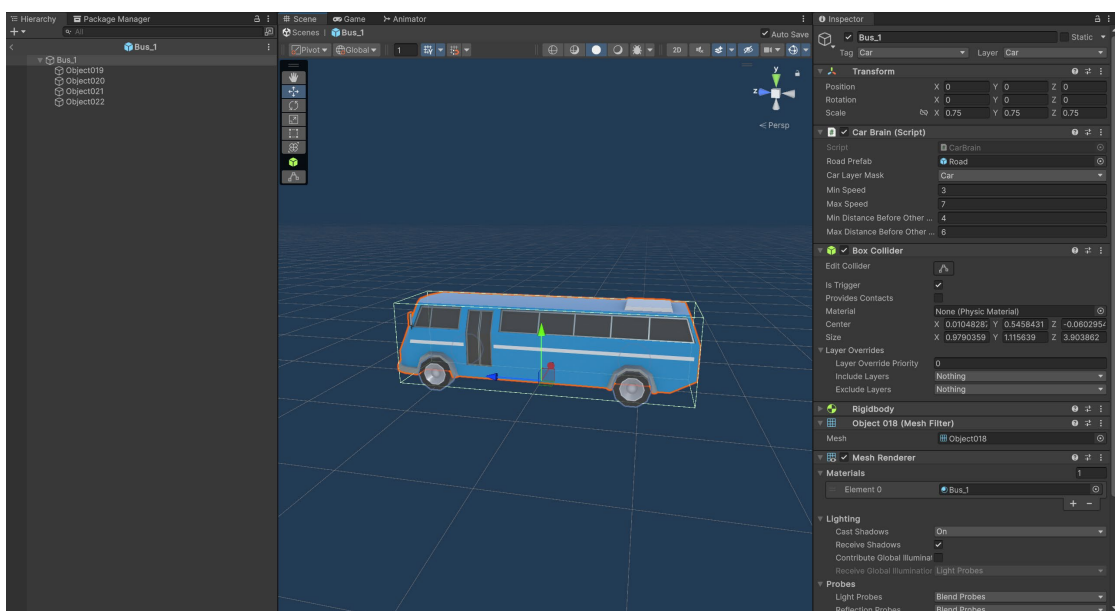


Рисунок 12 – Пример префаба автобуса

Так же для игры необходимы префабы с декорациями, которые помимо функции декоры будут также усложнять обучение для агента, делая обучение более похожим на реальную жизнь, так как данные объекты попадают на сенсор лучей, а больше объекты декора действуют как автомобили, приводя к

негативному завершению игры при столкновении. Примеры таких префабов представлены на рисунках 13 и 14.

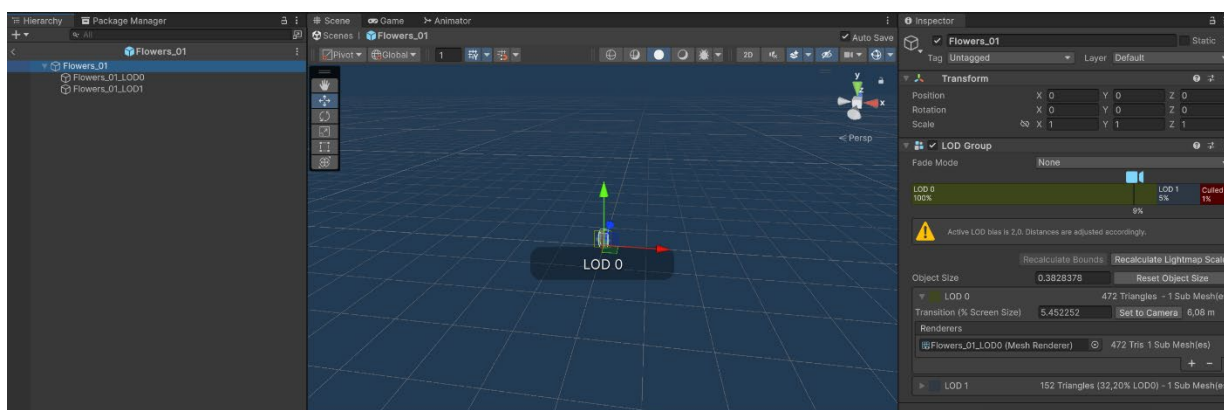


Рисунок 13 – Пример маленькой декорации (цветок)

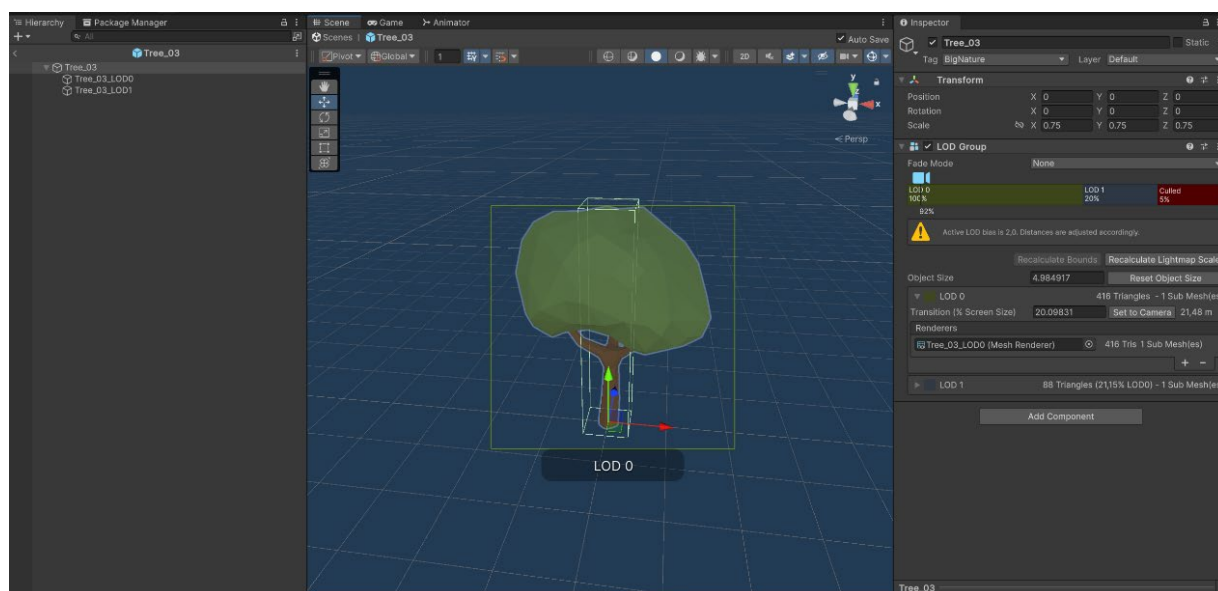


Рисунок 14 – Пример большой декорации (дерево)

Большинство моделей бралось из Unity Asset Store с изменениями и настройками под проект [27].

Далее рассмотрим логику работы машины и то, как она взаимодействует со средой. Для начала определим, что на игровой сцене, на всех дорогах машины будут перемещаться снизу-вверх. Дороги на сцене ограничены, и мы хотим, чтобы машина по достижению конца дороги перемещалась в её начало, это более эффективно, чем удаление и создание объекта машины заново. Здесь мы добавляем первый элемент случайности: у машин при создании и при перемещении в начало уровня будет обновляться скорость и максимальная

дистанция для впереди находящейся машины, это также добавит более живого окружения.

Класс, определяющий логику работы машины наследуется от встроенного в Unity класса `MonoBehaviour`. Начальная позиция дороги высчитывается из длины переданного в скрипт префаба дороги. Движение машины прописано в методе `Move`, который вызывается из внутреннего метода `FixedUpdate`, обеспечивающего вызов через фиксированное число кадров. Движение выглядит как перемещение машины вперед, в зависимости от скорости, если впереди нет других машин, что проверяется путём направления луча вперёд. Код для передвижения представлен в листинге 1.

Листинг 1 – Метод для передвижения машины

```
private void Move()
{
    Vector3 rayPosition = transform.localPosition;

    // Проверяем есть ли впереди другая машина, если да, то не
    двигаемся
    if (Physics.Raycast(rayPosition, transform.forward,
        _stopDistanceBeforeOtherCar, _carLayerMask))
    {
        return;
    }

    transform.Translate(Vector3.forward * _speed *
        Time.fixedDeltaTime);
}
```

Также как было написано выше необходимо перемещать машину, когда она достигает конца дороги, для этого поставим на сцене в конце дороги стенку с коллайдером, при соприкосновении с которым машины будет перемещаться за счёт встроенного метода `OnTriggerEnter`.

Полный листинг файла с кодом для логики работы машины представлен в приложение А.1.

В начале уровня эти машины необходимо генерировать случайным образом, для это на сцене будет специальный игровой объект `CarDirector`, который будет содержать класс `CarGenerator`, отвечающий за эту генерацию. Для скрипта сериализуются множество полей, для получения объектов машин, дороги,

настройки сцены, а также сложности в виде поля отвечающего за интенсивность трафика.

Для каждой дороги генерируются столько машин, сколько указано в интенсивности трафика, листинг 2 показывает метод генерации машин.

Листинг 2 – Метод для генерации машин

```
private void GenerateCars()
{
    float spawnXPosition = _firstRoad.transform.localPosition.x;

    float roadCenterZPostion =
_firstRoad.transform.localPosition.z;
    float roadStartZPosion = roadCenterZPostion - _roadLength /
2;
    float roadEndZPostion = roadCenterZPostion + _roadLength /
2;

    for (int i = 0; i < _roadNumber; i++)
    {
        for (int j = 0; j < _trafficIntensity; j++)
        {
            float spawnZPosition =
Random.Range(roadStartZPosion, roadEndZPostion);

            var car = Instantiate(_carPrefabs[Random.Range(0,
_carPrefabs.Count)]);

            car.transform.SetParent(transform);
            car.transform.localPosition = new
Vector3(spawnXPosition, _spawnYPosition,
spawnZPosition);
        }

        // Переходим на следующую дорогу
spawnXPosition += _roadWidth;

        // Пропускаем поле с травой
if (i + 1 == _roadsBeforeMiddleGrass)
{
            spawnXPosition += _middleGrassWidth;
        }
    }
}
```

Класс CarGenerator также наследуется от MonoBehaviour. Полный код для генератора машин можно увидеть в листинге А.2.

2.2 Подготовка агента к обучению

Следующим шагом является описание логики персонажа, учитывая необходимость реализовать его и как агента, способного обучаться и действовать после этого, и как персонажа под управлением игрока. Для этого в ML-Agents есть специальный класс, позволяющий реализовать всё это – Agent, наш класс PlayerAgentBrain будет от него наследоваться. Полностью данный класс представлен на листинге А.3.

Всё обучение будет состоять из эпизодов, для определения действий в начале эпизода достаточно переопределить метод OnEpisodeBegin, где мы устанавливаем случайный поворот, случайную позицию по координате z и рассчитываем расстояние до финиша, это можно увидеть на листинге 3.

Листинг 3 – Начало эпизода

```
public override void OnEpisodeBegin()
{
    float spawnZPosition = Random.Range(_spawnStartZPosition,
    _spawnEndZPosition);

    transform.SetLocalPositionAndRotation(
        new Vector3(_spawnXPosition, _spawnYPosition,
    spawnZPosition),
    Quaternion.Euler(0f, Random.Range(-180f, 180f), 0f));

    _lastDistance = _finishXPosition -
    transform.localPosition.x;
}
```

При обучении входные параметры автоматические подставляются в модель в зависимости от значений за которыми необходимо наблюдать. Так мы можешь добавить свои переменные, которые необходимо отслеживать с помощью переопределения метода CollectObservations, который показан в листинге 4.

Листинг 4 – Добавление значений для наблюдения.

```
public override void CollectObservations(VectorSensor
sensor)
{
    // Добавляем в наблюдение текущее значение расстояния
    до финиша
    sensor.AddObservation(_currentDistance);
    // Добавляем в наблюдение позицию агента - 3 координаты
    sensor.AddObservation(transform.localPosition);
    // Добавляем в наблюдение поворот агента - 4 компоненты
```

```

        Кватерниона
        sensor.AddObservation(transform.localRotation);
    }

```

Как описано в комментариях данного метода мы добавляем для наблюдения расстояние до финиша, позицию агента в мире (3 координаты) и поворот агента (4 компоненты), который представляет собой кватернион и описывается структурой Quaternion, которая встроена в Unity.

Для описания действий агента на каждом шаге переопределяется метод OnActionReceived, который принимает структуру буферов ожидаемых действий считывая эти действия мы можешь определять поведение агента. Так Первое значение последовательности представляет собой движения агента:

- 0 – не двигаться,
- 1 – идти вперед,
- 2 – идти назад.

Второе значение отвечает за поворот агента:

- 0 – повернуться налево,
- 1 – повернуться направо,
- 2 – не поворачиваться.

Также важным шагом для обучения будет добавление награды, если агент уменьшил рекордное минимальное расстояние до финиша, и небольшого накладывание штрафа, если агент этого не сделал, это позволяет сделать метод AddReward класса Agent, часть этого кода внутри метода OnActionReceived показана на листинге 5. Данное изменение награды является одним из ключевых моментов обучения, так как это показывает агенту какие его действия были правильными, а какие нет, и как нужно действовать стратегически (для этого штраф на отсутствие действий сделал очень маленьким).

Листинг 5 – Изменение награды в зависимости действий агента на каждом шаге

```

// Рассчитываем текущее расстояние до финиша
_currentDistance = _finishXPosition - transform.localPosition.x;

/* Сравниваем текущее расстояние до финиша с расстоянием на
предыдущем шаге.

```

```

* Если он меньше добавляем награду, иначе уменьшаем.
Пересчитываем расстояние
*/
if (_currentDistance < _lastDistance)
{
    AddReward(0.3f);
    _lastDistance = _currentDistance;
}
else
{
    AddReward(-0.001f);
}

```

Далее определим поведение персонажа при управлении игроком с клавиатурного ввода, для этого можно переопределить встроенный в класс Agent метод `Heuristic`, который имеет ссылочный параметр для действий агента, внутри данного метода мы задаём только значения для последовательностей действий персонажа в зависимости от нажатой на клавиатуре кнопки, а сами действия обрабатываются в зависимости от этих действий в описанном выше методе `OnActionReceived`.

Так же для персонажа нужно как-то отслеживать полное прохождение дороги и заход на объекты машины или больших объектов природы. Также необходимо добавить стены, как было сделано для машин. Только в случае машин была одна стена, при соприкосновении с которой их перемещало в начало уровня, для персонажа будет добавлено 4 стены, при соприкосновении с которыми игра (эпизод) будет завершаться. Это делается, чтобы ограничить пространство, в котором может перемещаться персонаж. Для удобства на финальную сторону дороги будет добавлен невидимый объект с коллайдером и все описанные выше взаимодействия будут отслеживаться в методе `OnTriggerEnter`, где при соприкосновении с машиной, концом дороги или большим объектом природы будет накладываться штраф и завершаться эпизод, а при соприкосновении с финальной точкой дороги добавляться награда и также заканчиваться эпизод.

При добавлении скрипта `PlayerAgentBrain` на игровой объект также автоматически добавляется компонент `Behavior Parameters`, он отображён на рисунке 15.

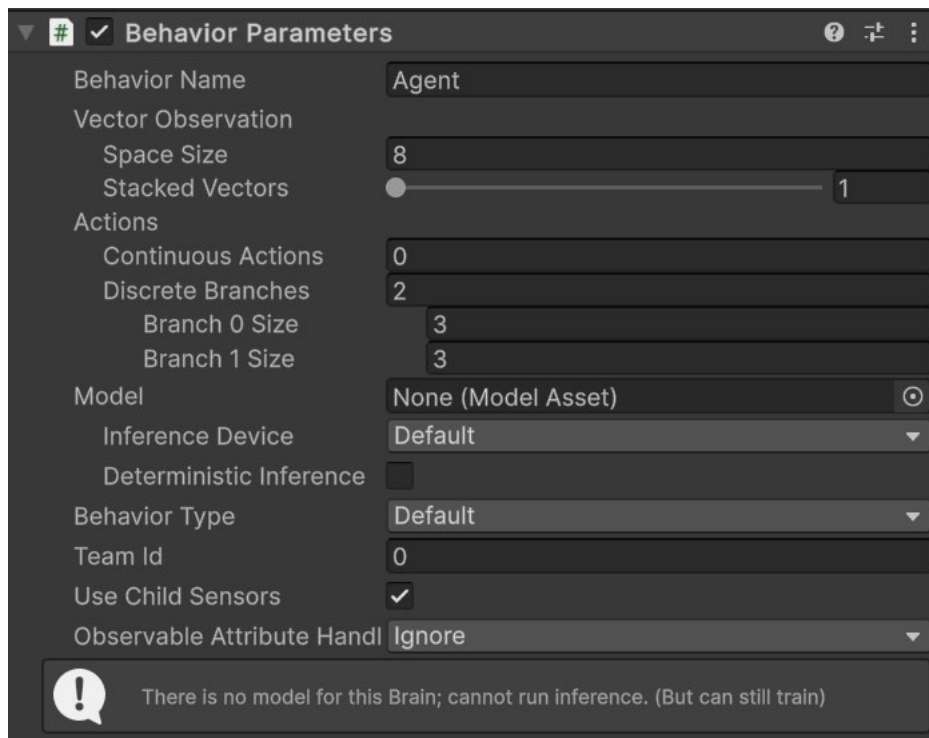


Рисунок 15 – Компонент агента Behavior Parameters

Рассмотрим важные поля данного компонента подробнее:

- Behavior Name – имя агента, которое используется при обучении,
- Vector Observation. Space Size – число дополнительных наблюдений, который добавляются через AddObservation. В нашем случае это значение равно восьми вышеописанным наблюдениям,
- Actions. Discrete Branches – указывает число отдельных действий за шаг, а Branch 0 Size и Branch 1 Size – количество возможных вариантов для этих действий. Все эти возможные действия описаны выше,
- Model – позволяет поставить для персонажа обученную модель, которая будет обрабатывать входные данные и как-то на них реагировать,
- Behavior Type – позволяет задать, будет ли агент выполнять действия модели или переопределённые действия в Heuristic. Значение Default позволяет одновременное учитывание действий модели и действий с клавиатурного ввода,
- Use Child Sensors – позволяет использовать сенсоры, которые есть у дочерних объектов этого агента. В нашем случае используется сенсор лучей, он будет представлен ниже [24].

У основного скрипта PlayerAgentBrain появляется поле Max Step, которое отвечает за максимальное число шагов, которые может сделать агент, после чего эпизод начнётся заново. Установим для этого поля значение 1000, слишком маленькое значение будет перезагружать эпизод, когда агент ещё будет стараться что-то сделать. Само это значение необходимо, чтобы начать эпизод заново, если агент где-то застрял и не сможет дойти в коде до вызова метода EndEpisode.

Также необходимо добавить компонент Decision Requester, он показан на рисунке 16. Данный компонент необходим для автоматического опроса решения от агента через определённый промежуток времени, который задан как период в Decision Period. Установленное значение для этого поля указывает через сколько академических шагов будет выполняться опрос. Также устанавливаем, что действие будут производиться между шагами принятия решения [24].

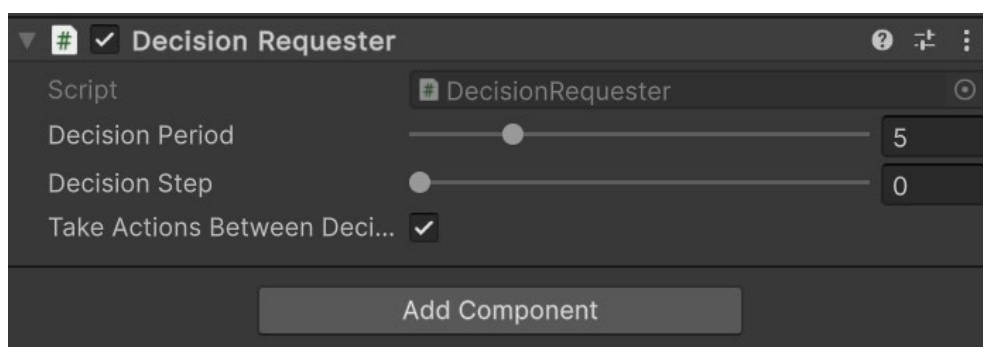


Рисунок 16 – Компонент Decision Requester

Как было описано выше для персонажа также добавляется объект на котором хранится компонент Ray Perception Sensor 3D, который входит ML-Agents. Данный компонент позволяет агенту «видеть», то есть он добавляет в модель наблюдения с помощью лучей. Данный компонент с подобранными настройками представлен на рисунке 17.

Игра и персонаж полностью подготовлены, уже сейчас можно играть в данную игру, управляю персонажем. Агент внутри среды также подготовлен к обучению, следующим шагом перед обучением будет настройка нейронной сети с определением архитектуры. Для ML-Agents эта настройка задаётся в yaml файлах, описание нейронной сети в таком формате представлено на листинге 6.

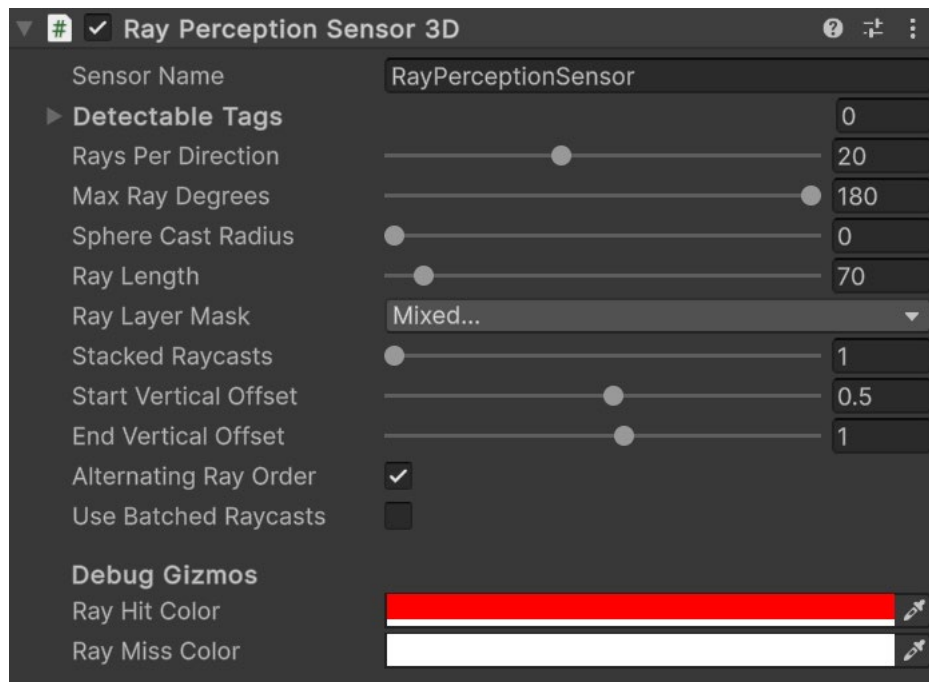


Рисунок 17 – Компонент Ray Perception Sensor 3D

Листинг 6. Пример настройки обучения агента в yaml

```
behaviors:
  Agent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 64
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2
      lambda: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 500000
    time_horizon: 1000
    summary_freq: 12000
```

Рассмотрим общие параметры данного файла подробнее:

– behaviors – что-то вроде ключевого слова, которое задаёт описание поведения;

– Agent представляет собой название поведения. Данное поле служит некоторым идентификатором, для обучения агента, необходимо, чтобы такое же название стояла у него в поле Behavior Name в компоненте Behavior Parameters;

– trainer_type – алгоритм, использующийся для обучения, наиболее часто используемые варианты при обучении с взаимодействием со средой: ppo (Proximal Policy Optimization) и sac (Soft Actor-Critic);

– hyperparameters – гиперпараметры сети:

1) batch_size – число опытов в каждой итерации градиентного спуска;

2) buffer_size – число опытов, который необходимо собрать перед обновлением модели политики. В SAC — максимальный размер буфера опытов;

3) learning_rate – начальная скорость обучения;

4) learning_rate_schedule – определяет изменения скорости обучения;

– network_settings – настройки нейронной сети:

1) normalize – указывает, нужно ли автоматически выполнять нормализацию наблюдений;

2) hidden_units – количество нейронов в скрытых слоях;

3) vis_encode_type – тип кодировщика для визуальных наблюдений. simple использует простой кодировщик из двух свёрточных слоёв;

– reward_signals – настройки сигналов награды:

1) gamma – коэффициент дисконтирования;

2) strength – коэффициент силы, на который умножается награда;

– keep_checkpoints – максимальное число контрольных точек модели, которое нужно хранить;

– max_steps – максимальное число шагов, которое нужно выполнить для завершения процесса обучения;

– time_horizon – указывает сколько шагов в % должен сделать агент, прежде чем буфер опыта;

– summary_freq – число опытов, которое нужно собрать для создания и отображения статистики [25, 28].

Выше описаны основные настройки, также для обучения могут использоваться дополнительные параметры, параметры специфичные для конкретных алгоритмов будут описаны ниже.

Игровая среда разработана с помощью Unity, взаимодействие с нейронными сетями происходит через пакет ML-Agents, при этом важно рассматривать персонажа, который находится на уровне как независимого, он может быть частью системы агента и контролироваться нейронной сетью, при этом не зависеть от конкретной нейронной сети, то есть она может меняться динамически. Также персонаж может быть игровым, под управлением человека. Разделение по компонентам в виде схемы представлено на рисунке 18.

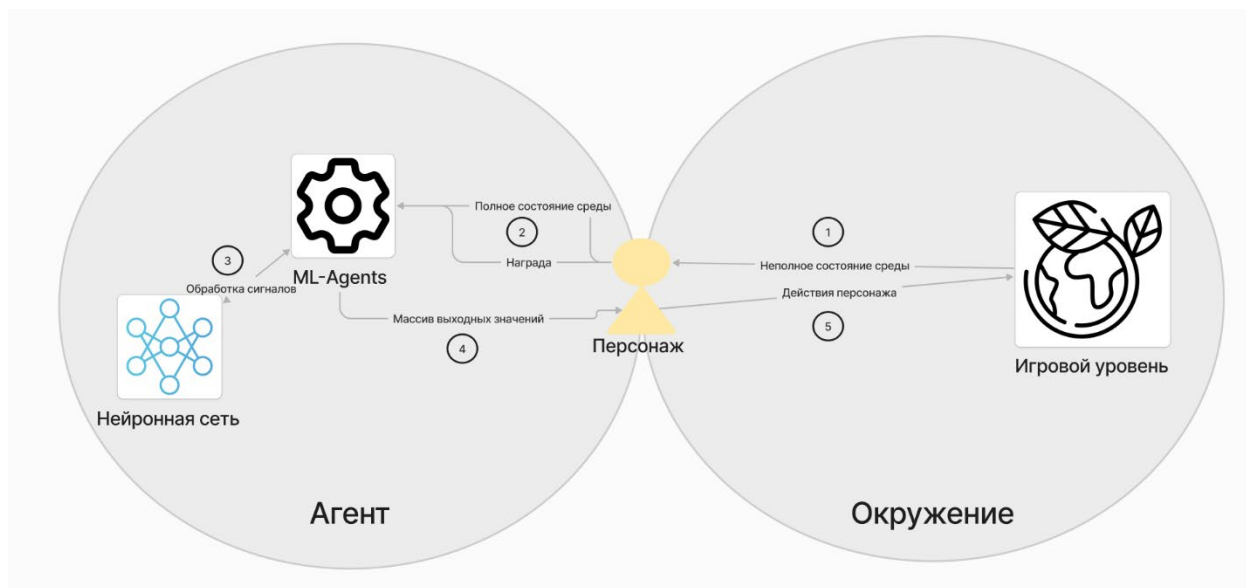


Рисунок 18 – Взаимодействие компонентов в система с агентом

В построенной системе агент и окружение взаимодействуют непосредственно через персонажа, который выступает частью как агента, так и среды. Частью среды он является потому, что считывает окружение в том виде, в котором это задано внутри него (в нашем случае, с помощью датчика лучей), а также может дополнять его (в данной работе, персонаж добавляет к считанным значениям свои расположение в мире, поворот, а также расстояние до цели), вместе с состоянием окружения именно внутри персонажа в зависимости от состояния среды определяется получаемая агентом награда. Частью системы агента персонаж является, так как он обрабатывает выходные сигналы нейронной сети и в зависимости от них выполняет определённые действия, которые

сказываются на среде. Также в этой системе есть отдельный компонент, который называется ML-Agents, он выступает посредником между персонажем и нейронной сетью, обеспечивая обучение по заданному алгоритму, сами алгоритмы будут рассмотрены ниже. Схема является общей как для взаимодействия при обучении, так и после того, как нейронная сеть уже обучена, но с некоторыми изменениями внутри шагов, сам процесс взаимодействия условно проходит в пять шагов и является итерационным, рассмотрим этот процесс. На первом шаге персонаж с помощью сенсора лучей считывает состояние среды каким его «видит». Далее персонаж дополняет считанное состояние своим положением, поворотом и расстоянием до цели, а также передаёт дальше награду или штраф, если условия для этого выполнены. На третьем шаге формируется ответ нейронной сети на переданное состояние среды. Помимо этого:

- если происходит этап обучения, то на основе предыдущего состояния среды, предыдущего ответа нейронной сети и полученной награды происходит изменение весов нейронной сети в зависимости от алгоритма, заданного в ML-Agents,

- если обучение не происходит, то полученная награда используется только для мониторинга внутри ML-Agents.

На четвертом шаге персонажу передаются только значимые для него выходные значения от нейронной сети, в данном случае это только 2 значения – для движения и для поворота. За заключительным шагом персонаж формирует действие в зависимости от полученных значений, оказывая влияние на среду.

Схема приведённая выше показывает взаимодействие компонентов в построенной системе, от нейронной сети можно было бы абстрагироваться и заключить её внутрь компонента ML-Agents, которому делегировано всё взаимодействие с ней, но данная схема вносит большей ясности, а также предполагает независимость нейронной сети от ML-Agents, так как после обучения мы получаем готовую нейронную сеть в виде файла с расширением .onnx и можем использовать его и с другой системой.

Если необходимо управлять агентом напрямую с устройств ввода, то данная возможность также имеется, прописана она через компонент ML-Agents, но может быть легко модифицирована под независимое использование, схема такого управления показана на рисунке 19.

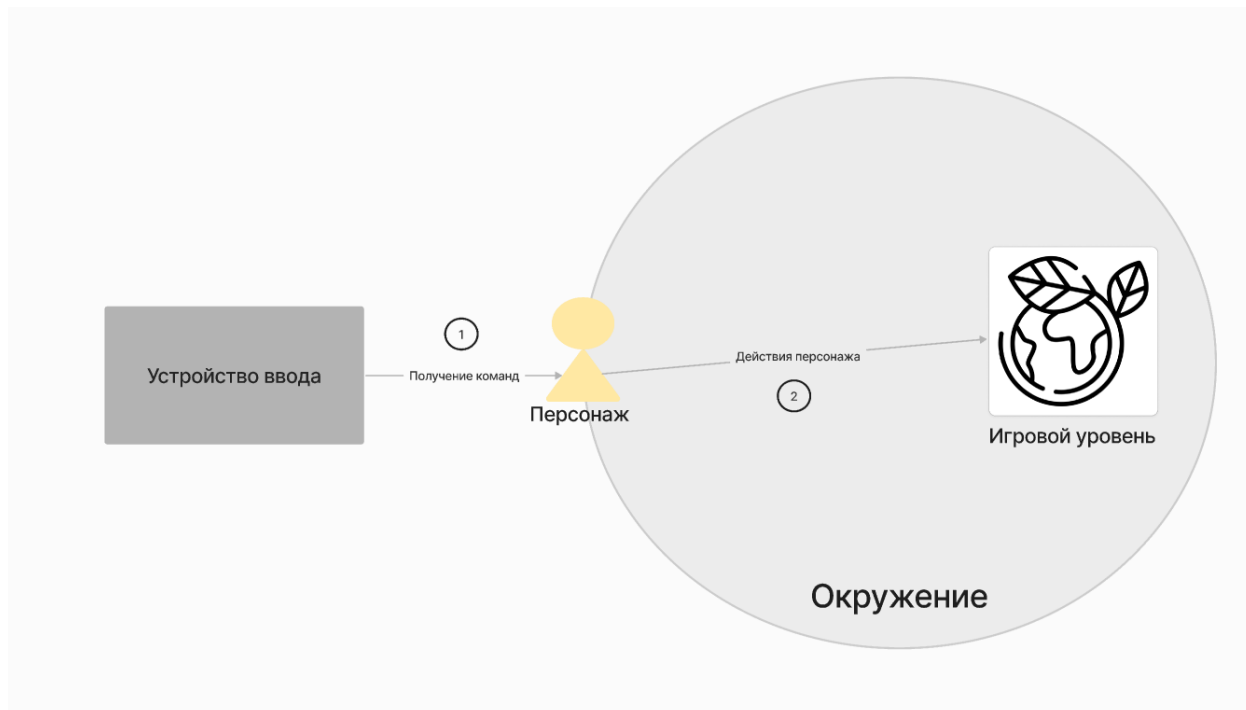


Рисунок 19 – Схема управления персонажем с устройств ввода

2.3 Описание алгоритмов обучения агента

Для обучения агента будет использоваться два алгоритма Proximal Policy Optimization и Soft Actor-Critic, их общие концепции были рассмотрены в первом разделе данной работы, рассмотрим их более подробно.

PPO представляет собой семейство методов первого порядка, которые стараются минимизировать функцию потерь при этом не сильно отходить от предыдущей политики. PPO использует нейронную сеть для аппроксимации идеальной функции, которая сопоставляет наблюдения агента с наилучшими действиями, которые агент может совершить при данном состоянии. Данный алгоритм используется через ML-Agent.

Выделяются PPO-Penalty и PPO-Clip, в основном, и в ML-Agents используется второй вариант, особенностью PPO-Clip является то, что данный алгоритм не использует никаких ограничений для KL(Kullback-Leibler)-дивергенции (относительная энтропия), но в целевой функции используются

специальные ограничения, которые предотвращают сильное изменение новой политики относительно старой. Функция потерь ($L^{CLIP}(\theta)$) для алгоритма PPO-Clip

$$L^{CLIP}(\theta) = \widehat{E}_t[\min(r_t(\theta)) \widehat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)) \widehat{A}_t], \quad (4)$$

где θ – параметр политики;

$\widehat{E}_t$ – эмпирическое ожидание на временных шагах;

r_t – отношение вероятностей новой и старой политики;

$\widehat{A}_t$ – предполагаемое преимущество на шаге t ;

ε – гиперпараметр обрезки (clipping) (обычно используется 0.1 или 0.2), который ограничивает изменения в политики.

Гиперпараметр ε – это то специальное ограничение, которое описано выше и которое применяется в операции обрезки. Оно определяет насколько сильно можно изменять политику на новом шаге, данный гиперпараметр имеет важное значение для стабильности и предсказуемости алгоритма [29, 30].

Общий алгоритм взаимодействия агента со средой описан в первом разделе данной работы, он остаётся неизменным. Рассмотрим параметры, конфигурирование которых предоставляется в ML-Agents:

- beta – сила энтропийной регуляризации. Данный параметр позволяет делать политику более случайной;

- epsilon – скорость развития политики во время обучения, данный коэффициент соответствует порогу расхождения между старой и новой политиками;

- lambda – параметр регуляризации лямбда, используется при расчёте GAE и указывает нужно агенту больше полагаться на фактическую награду среды или на текущую оценку;

– `num_epoch` – число эпох для выполнения оптимизации градиентного спуска;

– `shared_critic` – булево значение, указывающее нужно ли для функции политики и значения использовать одинаковый backbone (один фундамент для архитектуры) [25, 28].

Рассмотрим алгоритм Soft Actor-Critic, основная особенность алгоритмов из семейства Actor-Critic описана в первом разделе. Алгоритм SAC является off-policy алгоритмом, то есть он обучается на основе действий, полученных от другой политики, также SAC может извлекать данные для обучения из опыта, накопленного в прошлом. Опыт накапливается и помещается в специальный буфер воспроизведения опыта, оттуда он случайным образом извлекается и используется во время обучения, что делает данный алгоритм сэмпл эффективным.

Данный алгоритм исследует различные стратегии путём максимального разнообразия, которое делает Actor, за счёт того, что в SAC для Actor используется стохастическая политика, то есть для Actor на конкретном состоянии среды выбирается не конкретное оптимальное действие, а любое из возможных действий с определённой вероятностью, а цель обучения заключается в получении такого вероятностного распределения, чтобы получить максимально вознаграждение. Что бы использовать стохастическую политику в обучении агент получает дополнительную награду пропорциональную энтропии политики, то есть функцию награды добавляется энтропийная регуляризация.

В итоге, в SAC происходит одновременное обучение политики (того, что делает Actor) и двух Q-функций, параметризованных нейронной сетью, которые представляют собой двух Critic. Для данных Q-функций используется уравнение Беллмана, оно описано в первом разделе.

Общую работу данного алгоритма можно представить в виде пяти шагов:

1) при взаимодействии с окружением буфер воспроизведения опыта наполняется,

2) из буфера извлекается случайный набор данных,

- 3) исходя из текущей политики Actor определяется его действие,
- 4) при использовании моделей Actor и двух Critic прогнозируется стоимость будущего состояния,
- 5) модели обновляются [31, 32].

Специфичные для данного алгоритма параметры в ML-Agents:

– `buffer_init_steps` – число событий, которое необходимо собрать в буффер перед тем как обновлять модель политики. Данный параметр объясняется стохастической природой алгоритма SAC,

– `init_entcoef` – представляет собой начальный коэффициент энтропии и указывает объём изучаемой информации агентом в начале обучения. Влияет только на начальное значение энтропийного бонуса, чем меньше знает агент, тем более случайные действия пробует,

– `save_replay_buffer` – булево значение, указывающее нужно ли сохранять буфет воспроизведения опыта,

– `tau` – интенсивность обновления нейронной сети для Critic,

– `steps_per_update` – среднее отношение количества действий агента к количеству обновлений его политики. По умолчанию данный параметр равен 1 и это соответствует извлечению батча размером, указанным в `batch_size` и использованию его для обновления политики,

– `reward_signal_num_update` – количество шагов мини батча, которые необходимо сделать для обновления сигналов награды [25,28].

Следующим шагом будет непосредственно подготовка данных настроек и обучение агента. Для более быстрого обучения на сцене размещено 13 агентов. Визуальное представление процесса обучения агента в среде можно увидеть на рисунке 20.

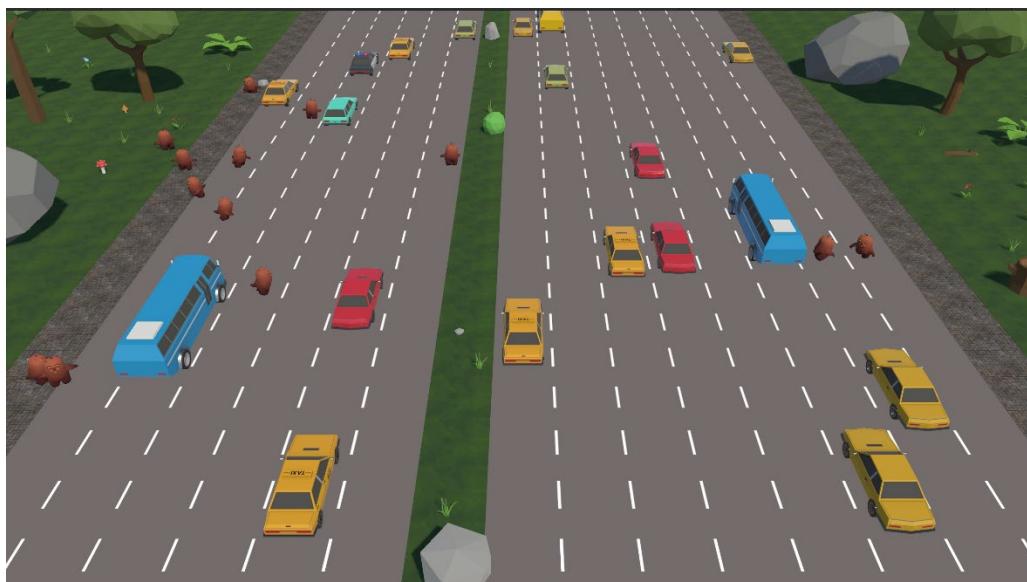


Рисунок 20 – Визуальное представление процесса обучения

При обучении агента подбирались различные гиперпараметры и настройки не только нейронной сети, но и настройки самого персонажа, реакции среды и так далее, соответственно обучение происходило множество раз, результаты процессов обучения будут представлены в разделе 2.4.

2.4 Анализ обучения агента

Обучение всех агентов происходило примерно в одинаковых условиях, на процессоре AMD Ryzen 7 7840H. При обучении агента подбирались различные гиперпараметры, настройки нейронной сети и агента в целом. По результатам многочисленных итераций обучения с различными настройками была сформирована базовая оптимальная настройка, дающая неплохие результаты, она будет описана ниже и именно с ней будут сравниваться нейронные сети с другими настройками.

Перед описанием базовой настройки рассмотрим систему наград. За удачное прохождение даётся 50 очков, за неудачную попытку отнимается 10, помимо этого агент как-то обучался и двигался к отмеченной цели вводится награда за продвижение агента вперёд, то есть, если расстояние агента до финиша стало меньше, чем предыдущее наименьшее расстояние, то ему начисляется награда 0.3 очка, если же агент движется в противоположную от финиша сторону, то у него отнимается 0.001 очка. Данный штраф сделал специально очень маленьким, что бы агент не боялся двигаться назад в экстренных случаях.

Базовые настройки, дающие неплохой результат представлены в листинге 7.

Листинг 7. Базовые настройки обучения агента в yaml

```
behaviors:
  Agent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size: 49152
      learning_rate: 1.0e-3
      beta: 0.001
      epsilon: 0.2
      lambda: 0.95
      num_epoch: 7
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
      vis_encode_type: resnet
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    checkpoint_interval: 300000
    max_steps: 3000000
    time_horizon: 128
    summary_freq: 5000
    threaded: true
```

В итоге, мы получаем с тремя входами, 5 выходами и с 31 слоем, большинство из них генерируются автоматически и не важны для рассмотрения. На вход подаётся массив `obs_0` с 82 значениями, это наблюдения агента, получаемые с сенсора лучей, массив `obs_1` имеет 8 значений, это дополнительные наблюдения, которые добавлялись в методе `CollectObservations`, а также на вход подаётся массив логических значений `action_masks`, имеющий 6 значений для маски каждого из действий, `true` указывает о недоступности действий на данном этапе обучения. На выходе получаем:

- `version_number` – версия ML-Agents для совместимости моделей,
- `memory_size` – размер памяти для хранения состояний агента,
- `discrete_actions` – количество дискретных действий агента,

– `discrete_action_output_shape` – форма выходного слоя нейронной сети, связанная с дискретными действиями,

– `deterministic_discrete_actions` – указывает, является ли выбор дискретных действий детерминированным. Если `true`, то будет выбираться действие с наивысшей вероятностью, иначе выбор действия будет на основе распределения вероятностей.

Большинство из этих значений явно не участвуют в обучении и нужны лишь для учёта особенностей. Так как нейронная сеть имеет довольно большой размер, всю её визуализацию разместить будет тяжело, но для наглядности начало нейронной сети представлено на рисунке 21, а конец на рисунке 22.

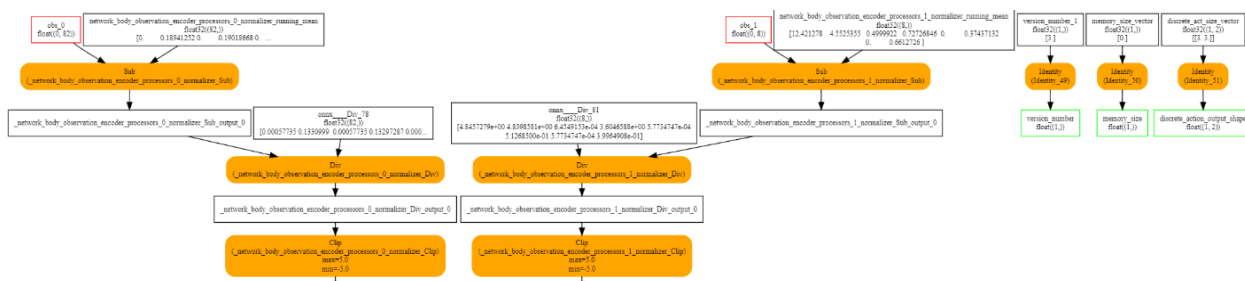


Рисунок 21 – Начало нейронной сети агента с базовыми настройками.

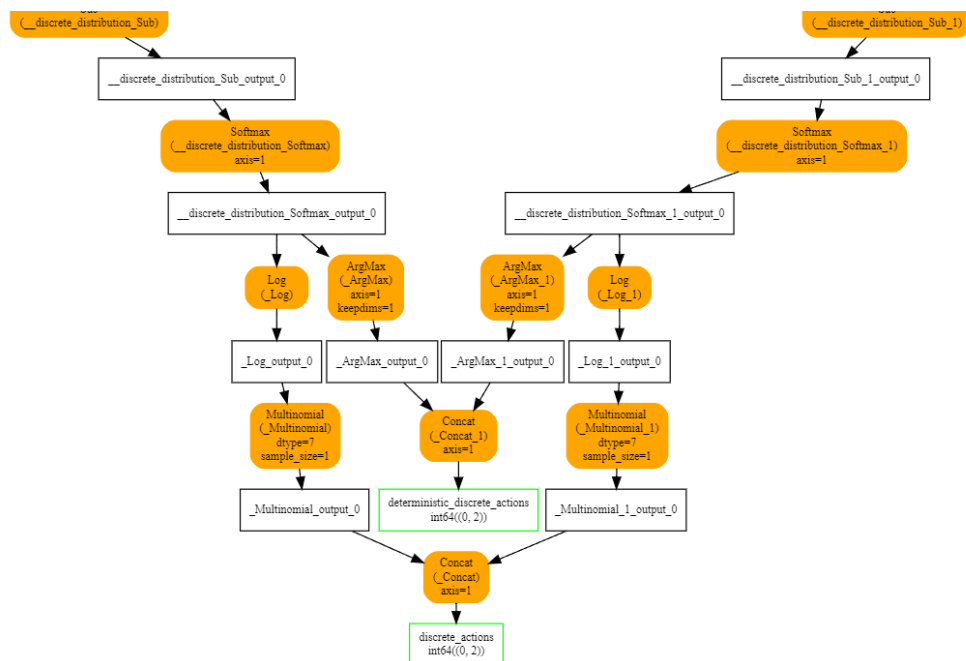


Рисунок 22 – Конец нейронной сети агента с базовыми настройками

Обучение с такими настройками заняло около 1 часа 20 минут. Проанализируем результаты процесса обучения, для начала рассмотрим кумулятивную награду, её график представлен на рисунке 23. Кумулятивная

награда показывает среднее кумулятивную награду за эпизод для всех обучающихся агентов на сцене. Значение кумулятивной награды должно увеличиваться в процессе тренировки, что будет указывать на её успешность. На графике видно постепенной увеличении данной функции, но также видно, что оно не стабилизировалось и можно продолжать обучение.

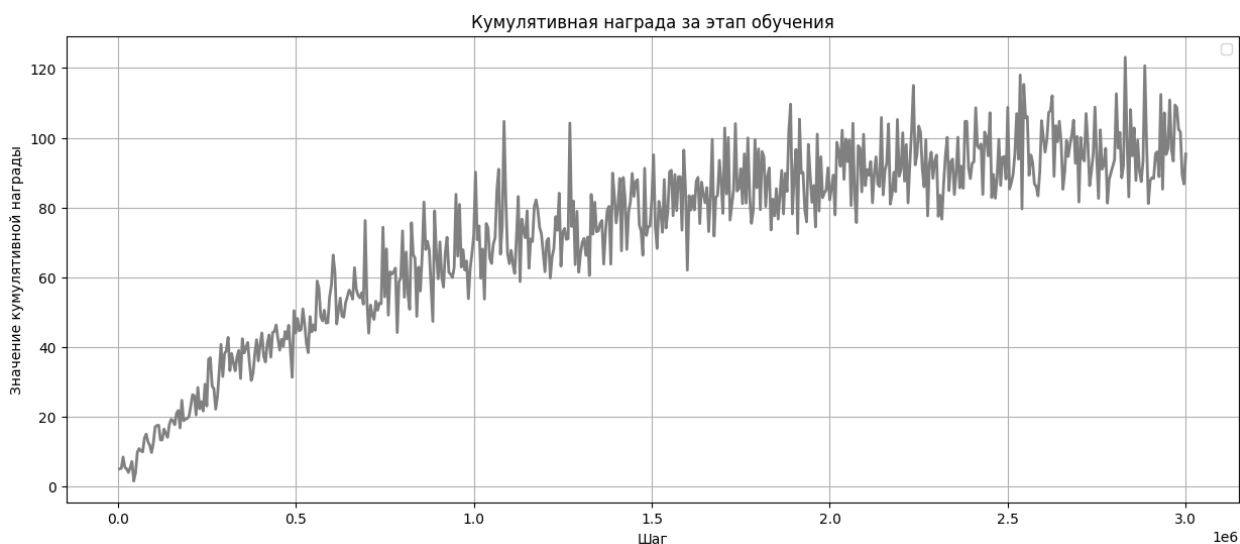


Рисунок 23 – График изменения кумулятивной награды в процессе обучения для базовых оптимальных настроек

Изменение потерь для policy-функции показано на рисунке 24. По оси у указана средняя величина функции потерь для policy на одном шаге для всех обучающихся агентов, она указывает изменение процесса принятия решения агентом, то есть на изменение его стратегии. При успешности тренировки данная величина должна уменьшаться, что видно и на графике.

Также важным будет рассмотреть изменения потерь value-функции, график этого изменения представлен на рисунке 25.

Данное значение при удачном обучении должно увеличиваться пока агент учиться, а затем уменьшаться, когда награда стабилизируется. На графике мы видим только увеличение, что подтверждает замечание, которое было сделано выше – агент ещё может продолжать обучение до стабильного состояния.



Рисунок 24 – График изменения потерь policy-функции в процессе обучения для базовых оптимальных настроек



Рисунок 25 – График изменения потерь value-функции в процессе обучения для базовых оптимальных настроек

В первую очередь рассматривалось количество задаваемых слоёв нейронной сети были рассмотрены варианты с разным количеством, но оптимальным было маленькое количество слоёв, для среды данного исследования 2 слоя давали наилучшие результаты. Так, например, нейронная сеть, настройки которой отличаются только количеством слоёв, которых установлено 7, обучается немного дольше по времени – примерно 1 час 39 минут, но при этом агент показывает результат хуже. График изменения кумулятивной награды для такого агента изображен на рисунке 26.

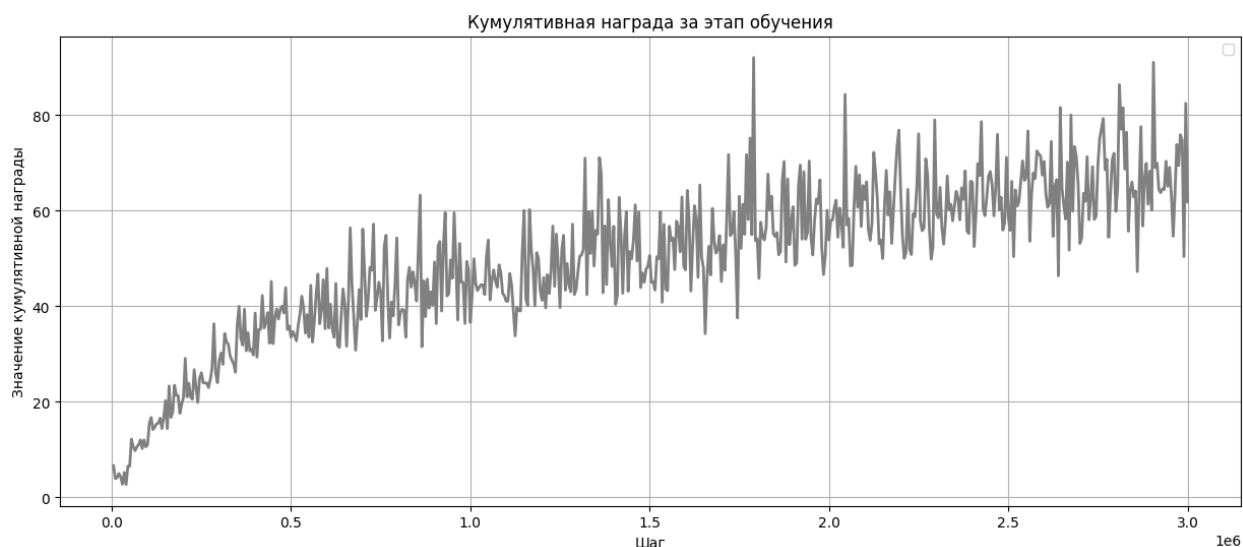


Рисунок 26 – График изменения кумулятивной награды в процессе обучения для сети с семью слоями

Графики функции потерь также показывают результаты хуже, чем с базовыми настройками, что указывает на более эффективное использование меньшего количества слоёв.

В начале этого раздела упоминалась системы награды агента, она оказывает очень большое влияние на обучение агента и подбору наград нужно уделить большое внимание при обучении агента. Нужно подобрать награду так, чтобы агент старался делать новые действия и как-то исследовать уровень, при этом он должен избегать опасных для него объектов, но это не стало для него настолько значимым, что он перестал изучать уровень вообще. Так, были оценены разные варианты награды для агента, на рисунке 27 показан график кумулятивной награды, при очень высоком штрафе за неудачной действие. Штраф был равен 30 очкам и это очень сильно повлияло на обучение, агент просто «боялся» продвигаться дальше.

Обучение агента с таким штрафом было закончено раньше, так как общая тенденция была видна, агент практически перестал обучаться, кумулятивная награда относительно стабилизировалось. Значение потери Value-функции изменялась очень слабо, на финальном шаге обучение – 2 070 000, она была равна 17.3

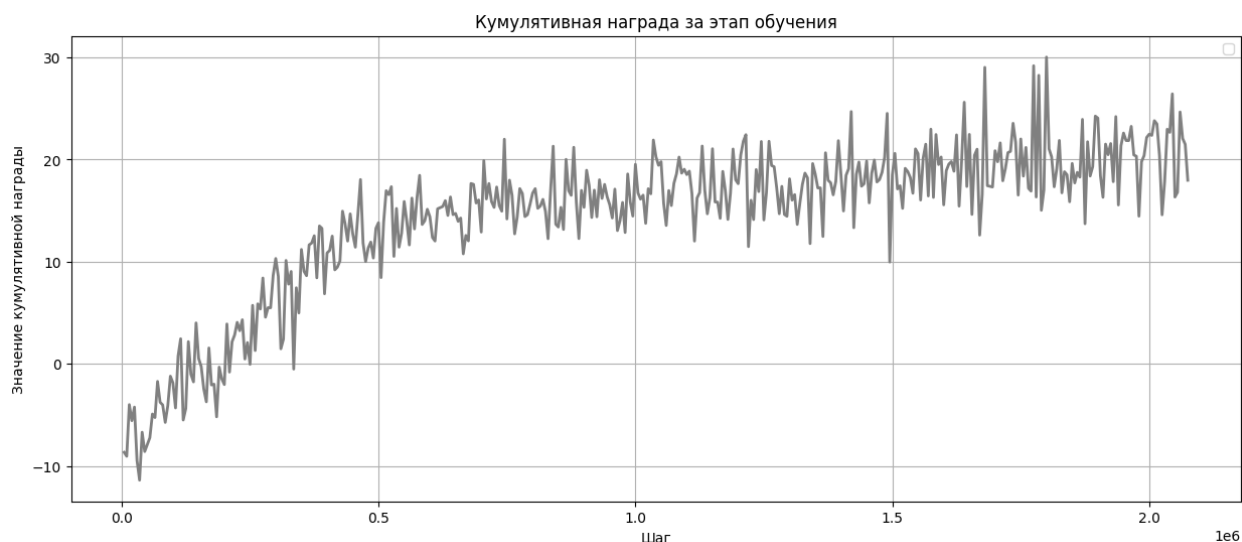


Рисунок 27 – График изменения кумулятивной награды в процессе обучения с высоким штрафом при неудаче

Такие же исследования были проведены для некоторых настроек сенсора лучей, те настройки, которые приведены на рисунке 17 оказались более удачны для отражения среды.

Данная среда создана с возможностью менять сложность, которая задаётся интенсивностью трафика. Все агенты до этого обучались на втором уровне сложности, который принят как основной в игре, но также были обучены агенты на первом и третьем уровнях с базовыми настройками, график кумулятивной награды для агента обученного на 1-ом уровне представлен на рисунке 28, а для агента на 3-ем уровне на рисунке 29.

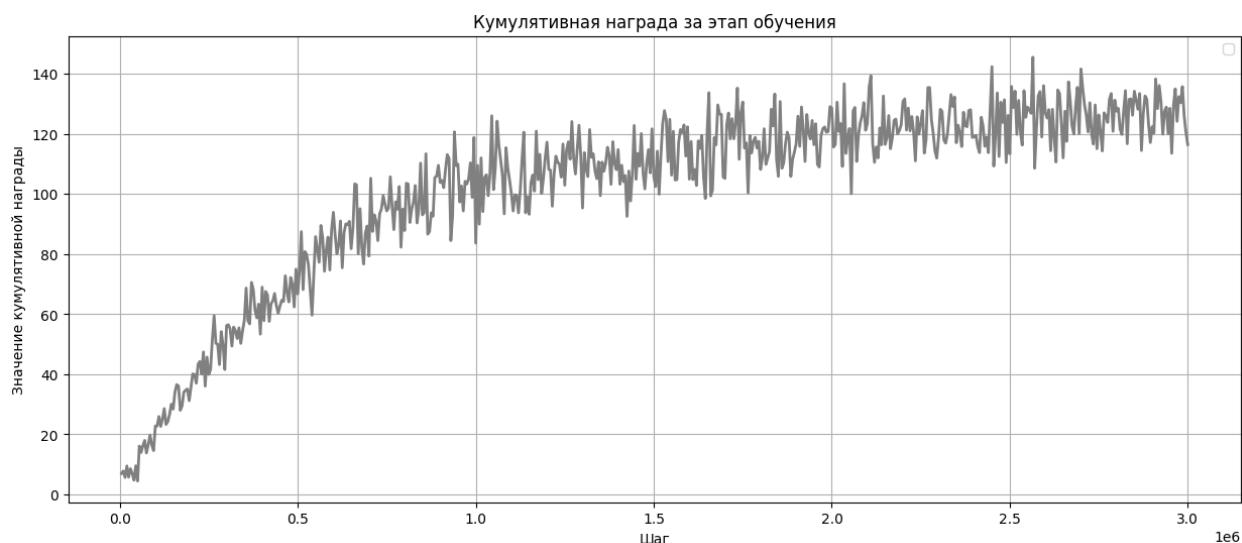


Рисунок 28 – График изменения кумулятивной награды в процессе обучения для агента, обученного на 1-ом уровне

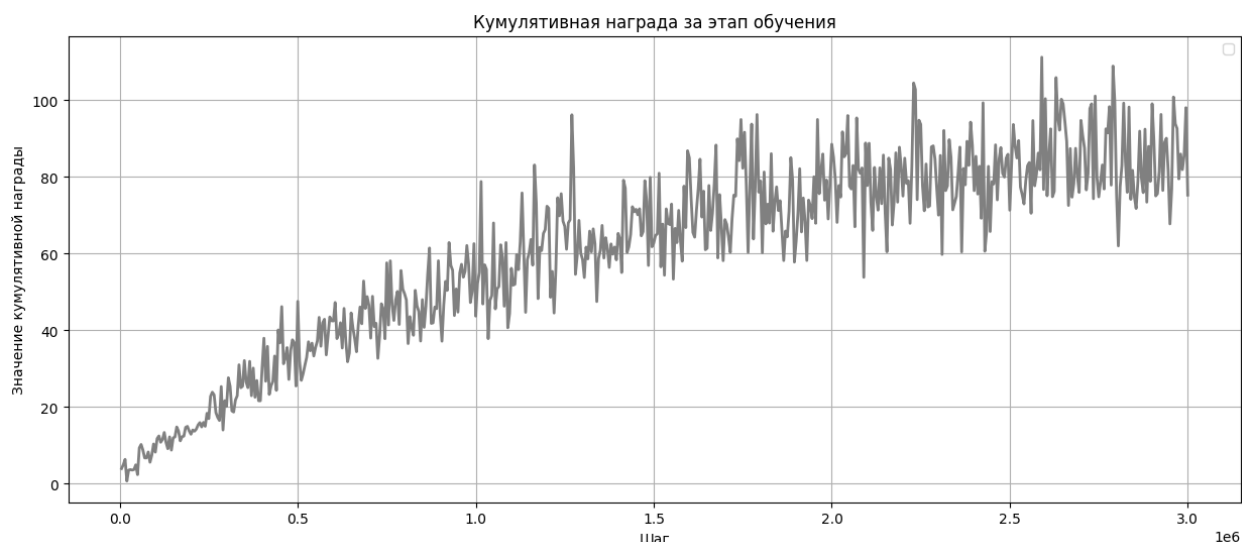


Рисунок 29 – График изменения кумулятивной награды в процессе обучения для агента, обученного на 3-ем уровне

Сравнивая 3 графика изменения кумулятивной награды, можно заметить, что чем проще уровень, тем получение награды происходит быстрее и как бы сам агент быстрее обучается, что логично и объясняет более простыми условиями. На графике потерь для value-функции у агента 1-го уровня также видно уменьшение значения, что указывает на то, что может указывать на то, что агент близок к концу завершения обучения. В целом, сравнение кумулятивной функции для этих случаев не совсем корректно, так как практически получаются различные среды, но можно сравнить как эти агенты уже после обучения проходят дорогу на разных уровнях сложности. Тестирование происходило в одинаковых условиях, на сцене присутствовало 12 агентов, тестировалось количество удачных и неудачных попыток за 1 минуту, данное сравнение приведено в таблице 1.

На первом уровне видно, что агента 3-го уровня справляется незначительно, но лучше других, однако уже на 2 уровне агент, который обучался на этом же уровня показывает результаты значительно лучше остальных, но наибольший интерес вызывают результаты 3-го уровня. Так, удачных попыток у всех почти одинаково, однако агенты 1-го и 2-го уровней значительно чаще ошибались, то есть агенты, обученные на менее суровых условиях вели себя более активно, что чаще приводило к неудачам. Агенты 3-го уровня лучше себя показали в таких условиях, однако слишком большая осторожности негативно сработала при игре на втором уровне сложности. Лучшим вариантом будет обучать агента в условиях

наиболее приближенных к тем, в которых он будет находиться. В некоторых ситуациях, например, когда игрок выбирает легкий уровень соперника может использоваться данный способ, чтобы не упрощать поведение агента, его можно сделать более осторожным. Также присутствует возможность дополнительно обучать агентов низкого уровня на более высоких уровнях при необходимости.

Таблица 1 – Сравнение попыток прохождения уровней разной сложности агентами, обученными на разных уровнях сложности

Уровень сложности среды	Результат попытки	Агент, обученный на 1-ом уровне	Агент, обученный на 2-ом уровне	Агент, обученный на 3-ем уровне
1	Удачно	54	51	61
	Неудачно	24	22	18
	Σ	78	73	79
2	Удачно	30	44	28
	Неудачно	50	37	49
	Σ	80	77	77
3	Удачно	13	14	12
	Неудачно	71	67	28
	Σ	84	81	40

Следующим шагом идёт анализ нейронной сети для работы с визуальными наблюдениями агента. Для этого в ML-Agents есть настройка `vis_encode_type` позволяющая задать несколько возможных вариантов. В базовой настройка указана сеть `resnet`, которая реализована как `IMPALA Resnet` и является самой большой из возможных вариантов. Также есть другие варианты:

- `simple`, используется 2 свёрточных слоя;
- `CNN`, реализации из статьи `Human-level control through deep reinforcement learning` от Мниха и других, которая состоит из 3х свёрточных слоёв.

Также есть другие варианты, но для обучения использовались эти три варианта и они не дали значительных отличий в результатах. Если среда имеет очень простые визуальные наблюдения и они точно не будут усложняться в будущем, то можно использовать более простые варианты, например настройку

simple, но в остальных случаях желательно использовать более сложные сети, как resnet.

Далее проанализируем результаты обучения с помощью алгоритма SAC, полные настройки для такого агента представлены на листинге 8.

Листинг 8. Настройки обучение агента алгоритмом SAC

```
behaviors:
  Agent:
    trainer_type: sac
    hyperparameters:
      learning_rate: 1.0e-4
      learning_rate_schedule: constant
      batch_size: 512
      buffer_size: 500000
      buffer_init_steps: 1000
      tau: 0.005
      steps_per_update: 10.0
      save_replay_buffer: false
      init_entcoef: 0.5
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.9
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 3000000
    time_horizon: 128
    summary_freq: 5000
    threaded: true
```

Обучение такого агента заняло значительно больше времени, чем агента с базовыми настройками, примерное время обучение 8 часов.

График изменения кумулятивной награды представлен на рисунке 30.

Видно, что кумулятивная награда, по сравнению с базовыми настройками на PPO алгоритме, стабилизировалась на более высоких значениях, что говорит о большей обученности агента, но и времени на обучение было потрачено значительно больше.

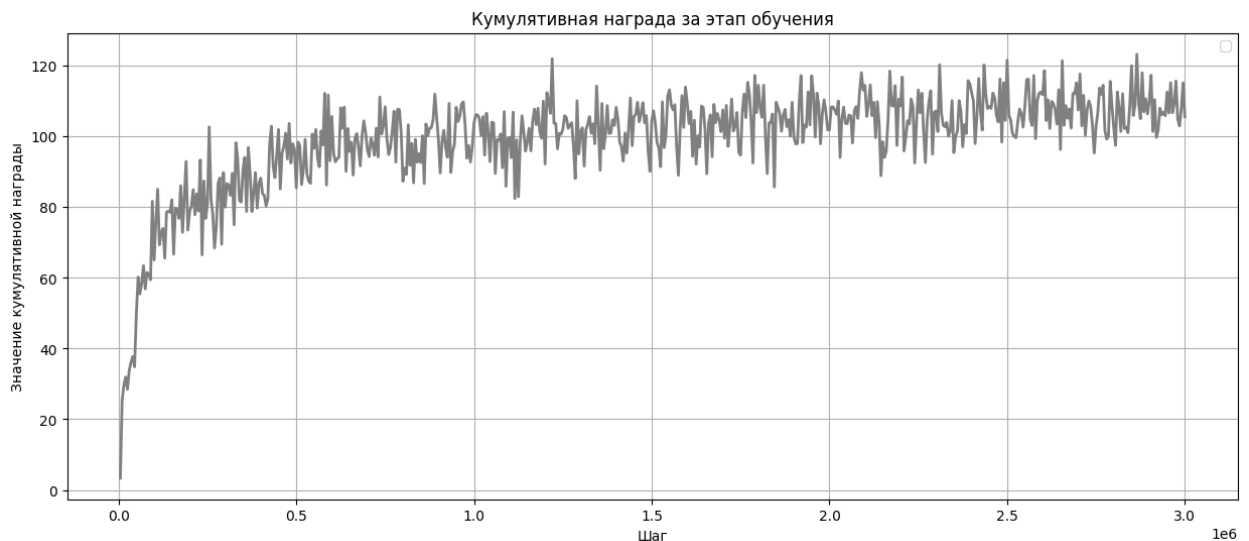


Рисунок 30 – График изменения кумулятивной награды в процессе обучения агента на алгоритме SAC

Дальше рассмотрим изменение потерь для policy-функции, график таких изменений показан на рисунке 31.

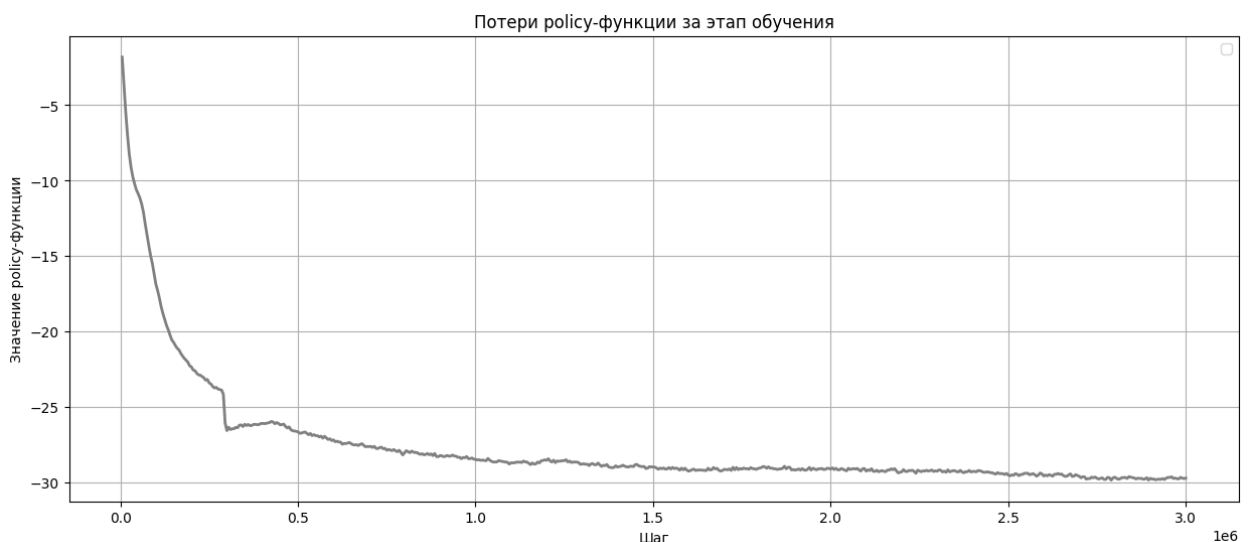


Рисунок 31 – График изменения потерь policy-функции в процессе обучения агента на алгоритме SAC

Видно, что обучение продолжает идти, но изменение потерь уже после 1 500 000 практически вышло на плато. Для более эффективной стратегии ещё можно было бы продолжать обучение, но оно уже не так эффективно, и при ограничениях по времени можно задать значительно меньшее число шагов.

Рассмотрим изменение потерь для value-функции, график таких изменений показан на рисунке 32.

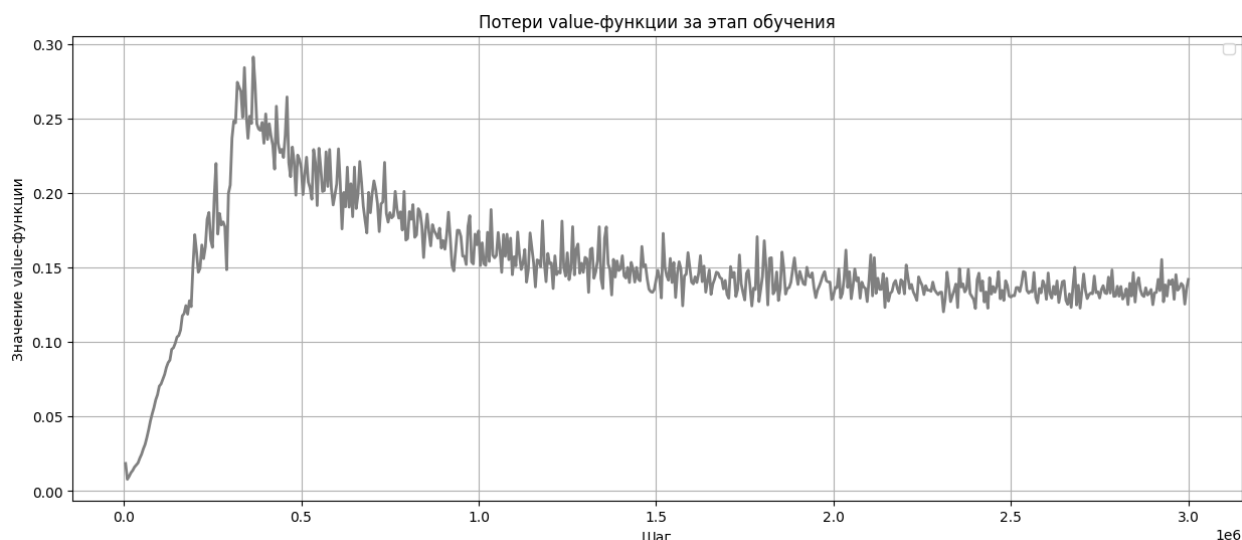


Рисунок 32 – График изменения потерь value-функции в процессе обучения агента на алгоритме SAC

Данный график также показывает относительную стабилизацию, но значения постепенно продолжает уменьшать, то есть активная фаза обучения прошла и агент уже продолжает обучаться деталям среды, это может быть полезно, если агента планируется использовать в четко заданной среде, но может негативно сказываться на универсальности.

SAC является более затратным алгоритмом, но при этом более эффективен в реальных средах, так как он не чувствителен к гиперпараметрам, их настройка сведена к минимуму благодаря энтропии. Также алгоритм позволяет использовать данные повторно, обеспечивая более эффективное обучение там, где нет возможности постоянно обучать агента непосредственно в среде. Данный алгоритм обеспечивает более простое возобновление обучения и устойчивость к шумам [32].

В более простых средах рекомендуется использовать алгоритм PPO как более простой и менее затратный, однако, если среда является достаточно сложной для стратегии агента и имеет много случайных факторов алгоритм SAC покажет более высокую эффективность, а вместе со случайными факторами большую универсальность агента. Таким образом, для большинства игровых сред подойдут настройки, которые были показаны выше и обозначены как базовые, они могут использоваться без изменений или выступать как baseline с подбором гиперпараметров для более эффективного обучения в конкретной среде.

2.5 Описание приложения

В предыдущих пунктах была разработана среда, обучен агент, но игра не доработана до конца. Первым делом необходимо среду на разные режимы.

Первый режим представляет собой одиночную игру, позволяя игроку управлять персонажем с клавиатурного ввода, цель режима – дойти до конца дороги. Оформленный одиночный уровень представлен на рисунке 33.



Рисунок 33 – Одиночный режим

Для персонажа используется тот же код, использовался выше при обучении агента, он представлен в приложении А.3, только в настройках агента отсутствует модель для управления и установлено эвристическое поведение.

Другой режим – «игрок против агента». Персонаж игрока повторяет персонажа из одиночной игры, а для агента используется модель, обученная на алгоритме SAC, настройки которого представлены выше в листинге 8. Скриншот данного режима показан на рисунке 34.

Следующий режим представляет собой соревновательную игру вдвоём (игрок против игрока). Для первого игрока используется тот же скрипт из одиночной игры, а для второго игрока создан новый скрипт, который повторяет

предыдущий, только считывает нажатие клавиш не W, S, A, D, как для первого игрока, а со стрелок $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ соответственно.



Рисунок 34 – Режим игрок против агента

Последний режим представляет собой не совсем игру: это просто уровень для просмотра за игрой агентов, на нём расположены 13 агентов, обученные с теми же настройками, что и агент в режиме «игрок против агента». Визуально это прохождение выглядит также как процесс обучения, который представлен на рисунке 20.

Также было оформлено меню, шаблон которого, с изменениями и доработками, был взят из Unity Asset Store [27]. Данное меню представлено на рисунке 35.

То что реализовано в игре на данный момент лишь небольшая часть того, что предполагалось реализовать изначально и игра может продолжать развиваться во многих направлениях: детальный выбор сложности с значительными изменениями на уровне, добавление бесконечного режима дороги, добавление множество связанных уровней с прогрессом и постепенным усложнением, добавление игры по сети, обучение более «умных» агентов, создание режима командной игры, когда одна команда пытается пройти дорогу, а другая ей мешает,

в том числе обучение агентов для игры в этот режим и множество других вариантов развития игры.



Рисунок 35 – Главное меню игры

То что реализовано в игре на данный момент лишь небольшая часть того, что предполагалось реализовать изначально и игра может продолжать развиваться во многих направлениях: детальный выбор сложности с значительными изменениями на уровне, добавление бесконечного режима дороги, добавление множество связанных уровней с прогрессом и постепенным усложнением, добавление игры по сети, обучение более «умных» агентов, создание режима командной игры, когда одна команда пытается пройти дорогу, а другая ей мешает, в том числе обучение агентов для игры в этот режим и множество других вариантов развития игры.

2.6 Тестирование

Цель данного тестирования – выявление отклонений от нормальной поведения игры. Тестирование будет проходить путём прохождения игры множество раз для выявления ошибок, чтобы проверить работоспособность игры в различных условиях. Результаты тестирования представлены в таблице 2.

Таблица 2 – Тестирование приложения

№	Назначение	Ожидаемый результат	Полученный результат	Итог
1	2	3	4	5
1	Проверка способности персонажа передвигаться и поворачиваться	Персонаж движется и поворачивается в правильном направлении, анимация работает корректно	Персонаж правильно двигался и поворачивался в зависимости от нажатых клавиш, анимация отрабатывает корректно	Пройден
2	Проверка неудачности прохождения уровня	При попадании на опасный объект персонаж появляется в начале уровня	После попадания на любой из опасных объектов персонаж заново начинал уровень	Пройден
3	Проверка успешного прохождения уровня	Когда персонаж достигает конца дороги уровень успешно завершается, переносом его в начало	Персонаж достигает конца дороги и переносится в начало	Пройден
4	Агент пытается пройти дорогу	В режимах просмотра игры агентов или в режиме против агента он движется в сторону прохождения дороги, пытаясь обогать препятствия	Агент пытался проходить дорогу, выполняя как удачные, так и неудачные попытки	Пройден
5	В режиме «Игрок против игрока» оба персонажа управляются одновременно	Оба персонажа могут управляться независимо друг от друга. Для них выполняются тесты 1, 2 и 3	Оба персонажа независимо друг от друга могли корректно передвигаться и поворачиваться, а также выполнялись условия неуспешного и успешного прохождения уровня	Пройден

Продолжение таблицы 2

1	2	3	4	5
6	Корректная работа меню	Режимы в главном меню можно переключать, как с помощью стрелок клавиатуры, так и с помощью курсора мыши. Выбор уровня доступен через нажатие мышкой на кнопку уровня или через клавишу Enter.	Режимы в меню переключались как мышкой, так и с клавиатуры с помощью стрелок и клавиш A и D. Уровень включался при нажатии мышкой на кнопку или при нажатии клавиши Enter	Пройден
7	Корректная работа выхода	Можно выйти в главное меню из любого уровня. Из главного меню можно закрыть игру.	Из любого из режимов успешно осуществлялся выход в главное меню с помощью клавиши Escape, выход из игры осуществлялся при нажатии Escape в главном меню или при выборе соответствующей кнопки	Пройден

Из результатов тестирования видно, что игра полностью работоспособна и соответствует требованиям.

2.7 Выводы

В данном разделе были спроектировано и разработано окружение, агенты подготовлены к обучению, подробно описаны алгоритмы, которые используются в обучении, рассмотрено как взаимодействуют компоненты спроектированной системы. Также агенты были обучены на разных настройках, полученные результаты обучения проанализированы, подобраны базовые настройки для обучения агента, которые могут быть использованы в других средах, игра полностью разработана и протестирована.

ЗАКЛЮЧЕНИЕ

Во время исследования и анализа предметной области были рассмотрены различные варианты реализации ИИИ, в том числе обучение с подкреплением и нейронный сети, были подробно рассмотрены алгоритмы глубокого обучения, такие как Proximal policy optimization и Soft Actor-Critic. Для создания игры с реализацией ИИИ были проанализированы различные варианты, которые позволяют эффективно, с необходимым контролем и гибкостью разработать спроектированную систему.

В рамках данной выпускной квалификационной работы была разработана игра с внедрением игрового искусственного интеллекта для обеспечения поведения персонажа. В качестве игрового движка использовался Unity, а в основу ИИИ легло глубокое обучение с подкреплением, для реализации которого использовался пакет ML-Agents. Помимо разработанного приложения были исследованы варианты обучения агента с их анализом, с выделением у них достоинств и недостатков, были описаны рекомендации для их использования, а также были составлены базовые настройки для обучения агента, которые могут быть использованы для обучения в совершенно разных средах, как полностью готовый вариант, так и как baseline, то есть первоначальная настройка, относительно которой будут подбираться гиперпараметры для более эффективного обучения.

Полученные результаты обладают научной ценностью, которая заключается в перспективе дальнейших исследований в области обучения с подкреплением, развитие которой позволит внедрять агентов во все области человеческой жизнедеятельности.

Сама игра, независимо от описанных выше исследований, может использоваться как готовый продукт, для неё уже разработано несколько режимов, позволяющих играть одному, против агента или против друга. Разработанная игра имеет большой потенциал для дальнейшего развития, которое может включать, например, внедрение игры по сети, разработку уровней с прогрессом, в том числе случайно генерируемых, так же возможность добавления

новых режимов, позволяющих играть командами, для чего также могут использоваться агенты и много других направлений развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Боровская, Е. В. Основы искусственного интеллекта / Е. В. Боровская, Н. А. Давыдова. – 4-е изд. – М.: Лаборатория знаний, 2020. – 130 с.
2. Джоши, Патрик Искусственный интеллект с примерами на Python / Патрик Джоши. – СПб.: ООО "Диалектика", 2019. – 448 с.
3. Казиев, А. Б. Прикладное применение средств искусственного интеллекта в разработке компьютерных игр / А. Б. Казиев, С. Ю. Прокопюк // Международный студенческий научный вестник. – 2015. – № 3 (часть 2). – С. 228-230.
4. Игровой искусственный интеллект / Кушнир, В. Н, Кушнир [и др.] // Научные труды КубГТУ. – 2016. – № 2. – С. 149-158.
5. Буковшин, В. А. Интеллектуальные системы в компьютерных играх. Перспективы развития искусственного интеллекта в игровой индустрии / В. А. Буковшин, С. Г. Воскобойников // Современные материалы, техника и технологии. – 2017. – № 3 (11). – С. 21-36.
6. Паласиос, Х. Unity 5.x. Программирование искусственного интеллекта в играх / Х. Паласиос. – М.: ДМК Пресс, 2017. – 272 с.
7. Моделирование поведения агентов для реализации игрового искусственного интеллекта / А. О. Анохин, Н. П. Садовникова, А. В. Катаев, Д. С. Парыгин // Прикаспийский журнал: управление и высокие технологии. – 2020. – № 2 (50). – С. 85-99.
8. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 2-е изд. – М.: Издательский дом "Вильямс", 2007. – 1408 с.
9. Хайкин, Саймон Нейронные сети: полный курс / Саймон Хайкин. – 2-е изд. – М.: Издательский дом "Вильямс", 2006. – 1104 с.
10. Шампандар, Алекс Дж. Искусственный интеллект в компьютерных играх: как обучить виртуальные персонажи реагировать на внешние воздействия / Алекс Дж. Шампандар. – М.: ООО "И.Д. Вильямс", 2007. – 768 с.
11. Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. – 2-е изд. – М.: ДМК Пресс, 2020. – 552 с.

12. Welcome to Spinning Up in Deep RL! // OpenAI. Spinning Up URL: <https://spinningup.openai.com/en/latest> (дата обращения: 10.04.2024).
13. Deep Reinforcement Learning: Pong from Pixels // Andrej Karpathy blog URL: <https://karpathy.github.io/2016/05/31/rl/> (дата обращения: 13.04.2024).
14. Тимошкин М.О., Романова Е.Г. Применение метода глубокого Q-обучения для обучения агентов в логических играх // Математическое и компьютерное моделирование естественно-научных и социальных проблем. – Пенза: Пензенский государственный университет, 2022. – С. 199-206.
15. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. Playing Atari with Deep Reinforcement Learning, 2013.
16. Крон, Джон Глубокое обучение в картинках. Визуальный гид по искусственному интеллекту / Джон Крон, Грант Бейлелевельд, Бассенс Аглаэ. – СПб.: Питер, 2020. – 400 с.
17. Ведяхин, А. Сильный искусственный интеллект: на подступах к сверхразуму / А. Ведяхин [и др.]. – М.: Интеллектуальная Литература, 2021. – 323 с.
18. Пискарев Д.И., Разинков К.Ю. Искусственная нейронная сеть. Области применения и принцип работы // EUROPEAN RESEARCH. Сборник статей XV Международной научно-практической конференции: в 2 ч. Том. Часть 1. – Пенза: Наука и Просвещение (ИП Гуляев Г.Ю.), 2018. – С. 33-36.
19. Соснин А.С., Сулова И.А. Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLU, tahn // Наука. Информатизация. Технологии. Образование. Материалы XII международной научно-практической конференции. – Екатеринбург: Российский государственный профессионально-педагогический университет, 2019. – С. 237-246.
20. Иванько А.Ф., Иванько М.Ф., Сизова Ю.А. Нейронные сети: общие технологические характеристики // Научное обозрение. Технические науки. – 2019. – №2. – С. 17-23.

21. Щегольков М.В., Зинкин С.А. Свёрточная нейронная сеть в обучении с учителем для определения поведения игрового агента // Студенческий вестник. – 2023. – №16-12 (255). – С. 68-69.
22. Богатырева А.А., Виноградова А.Р., Тихомирова С.А. Исследование способности к Transfer learning сверточных нейронных сетей, обученных на Imagenet // Международный журнал прикладных и фундаментальных исследований. – 2019. – №7. – С. 106-111.
23. Gym Documentation URL: <https://www.gymlibrary.dev> (дата обращения: 15.05.2024).
24. Unity Documentation URL: <https://docs.unity3d.com> (дата обращения: 05.05.2024).
25. ML-Agents // Github URL: <https://github.com/Unity-Technologies/ml-agents> (дата обращения: 15.05.2024).
26. Anaconda URL: <https://www.anaconda.com> (дата обращения: 15.05.2024).
27. Unity Asset Store URL: <https://assetstore.unity.com> (дата обращения: 20.04.2024).
28. Training Configuration File // Unity ML-Agents Toolkit URL: <https://unity-technologies.github.io/ml-agents/Training-Configuration-File/> (дата обращения: 07.05.24).
29. Proximal Policy Optimization // OpenAI. Spinning UP URL: <https://spinningup.openai.com/en/latest/algorithms/ppo.html> (дата обращения: 06.05.24).
30. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal Policy Optimization Algorithms, 2017
31. Soft Actor-Critic // OpenAI. Spinning UP URL: <https://spinningup.openai.com/en/latest/algorithms/sac.html> (дата обращения: 06.05.24).
32. Haarnoja T., Pong V., Hartikainen K., Zhou A., Dalal M., Levine S. Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots, 2018

ПРИЛОЖЕНИЕ А

(рекомендуемое)

Исследование стратегий в компьютерных играх с применением методов
искусственного интеллекта

Листинг программы

Листинг А.1 – Файл «CarBrain.cs»

```
using UnityEngine;

public class CarBrain : MonoBehaviour
{
    [SerializeField]
    private GameObject _roadPrefab;
    [SerializeField]
    private LayerMask _carLayerMask;
    [SerializeField]
    private float _minSpeed = 4f;
    [SerializeField]
    private float _maxSpeed = 6f;
    [SerializeField]
    private float _minDistanceBeforeOtherCar = 2f;
    [SerializeField]
    private float _maxDistanceBeforeOtherCar = 4f;

    private const string _endRoadTag = "CarEndRoad";

    private float _speed;
    private float _stopDistanceBeforeOtherCar;
    private float _roadStartZPostion;

    private void Start()
    {
        float roadLength = _roadPrefab.transform.localScale.z;
        _roadStartZPostion = _roadPrefab.transform.position.z -
            roadLength / 2;

        RandomizeCarSettings();
    }

    private void FixedUpdate()
    {
        Move();
    }

    private void OnTriggerEnter(Collider trigeeredCollider)
    {
        if (trigeeredCollider.CompareTag(_endRoadTag))
        {
            CarRespawn();
        }
    }

    /// <summary>
    /// Обновление параметров машины.
    /// </summary>
    private void RandomizeCarSettings()
    {
        _speed = Random.Range(_minSpeed, _maxSpeed);
    }
}
```

```

        _stopDistanceBeforeOtherCar = Random
        .Range(_minDistanceBeforeOtherCar, _maxDistanceBeforeOtherCar
        );
    }

    /// <summary>
    /// Отвечает за движение машины вперёд.
    /// </summary>
    private void Move()
    {
        Vector3 rayPosition = transform.localPosition;

        // Проверяем есть ли впереди другая машина, если да, то
        не двигаемся
        if (Physics.Raycast(rayPosition, transform.forward,
        _stopDistanceBeforeOtherCar, _carLayerMask))
        {
            return;
        }

        transform.Translate(Vector3.forward * _speed *
        Time.fixedDeltaTime);
    }

    /// <summary>
    /// Респаун машины в начальной позиции дороги.
    /// </summary>
    private void CarRespawn()
    {
        transform.localPosition = new
        Vector3(transform.localPosition.x,
        transform.localPosition.y, _roadStartZPostion);

        RandomizeCarSettings();
    }
}

```

Листинг А.2 – Файл «CarGenerator.cs»

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

public class CarGenerator : MonoBehaviour
{
    [SerializeField]
    private List<GameObject> _carPrefabs = new();
    [SerializeField]
    private GameObject _firstRoad;
    [SerializeField]
    // Количество машин на 1 дорогу
    private int _trafficIntensity = 2;
    [SerializeField]
    private int _roadNumber = 14;
    [SerializeField]
    private int _roadsBeforeMiddleGrass = 6;
    [SerializeField]
    private int _middleGrassWidth = 1;

    private float _roadWidth;
    private float _roadLength;

    private const float _spawnYPosition = 0.5f;

    private void Awake()
    {
        _roadWidth = _firstRoad.transform.lossyScale.x;
        _roadLength = _firstRoad.transform.lossyScale.z;

        GenerateCars();
    }

    /// <summary>
    /// Создание машин на дорогах уровня.
    /// </summary>
    private void GenerateCars()
    {
        float spawnXPosition =
            _firstRoad.transform.localPosition.x;

        float roadCenterZPostion =
            _firstRoad.transform.localPosition.z;
        float roadStartZPosion = roadCenterZPostion -
            _roadLength / 2;
```

```

        float roadEndZPostion = roadCenterZPostion +
        _roadLength / 2;

        for (int i = 0; i < _roadNumber; i++)
        {
            for (int j = 0; j < _trafficIntensity; j++)
            {
                float spawnZPosition =
                Random.Range(roadStartZPosion, roadEndZPostion);

                var car =
                Instantiate(_carPrefabs[Random.Range(0,
                _carPrefabs.Count)]);

                car.transform.SetParent(transform);
                car.transform.localPosition = new
                Vector3(spawnXPosition, _spawnYPosition,
                spawnZPosition);
            }

            // Переходим на следующую дорогу
            spawnXPosition += _roadWidth;

            // Пропускаем поле с травой
            if (i + 1 == _roadsBeforeMiddleGrass)
            {
                spawnXPosition += _middleGrassWidth;
            }
        }
    }
}

```

Листинг А.3 – Файл «PlayerAgentBrain.cs»

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

/// <summary>
/// Класс реализующий поведение и логику обучения агента
/// </summary>
/// <remarks>
/// Действия агента представлены в виде последовательности int:
///     1-ое значение последовательности представляет собой
движения агента:
///     0 - не двигаться;
///     1 - идти вперёд;
///     2 - идти назад.
///     2-ое значение отвечает за поворот агента:
///     0 - повернуться налево;
///     1 - повернуться направо;
///     2 - не поворачиваться.
/// </remarks>
public class PlayerAgentBrain : Agent
{
    [SerializeField]
    private float _moveSpeed = 3f;
    [SerializeField]
    private float _turnSpeed = 5f;
    [SerializeField]
    private GameObject _spawnPlatform;
    [SerializeField]
    private GameObject _finishPlatform;

    private const float _spawnYPosition = 0.5f;
    private const string _runAnimationParametr = "isRunning";
    private const string _finishPlatformTag = "FinishPlatform";
    private const string _playerEndRoadTag = "PlayerEndRoad";
    private const string _carTag = "Car";
    private const string _bigNatureTag = "BigNature";

    private Animator _animator;
    private float _lastDistance;
    private float _currentDistance;
    private float _spawnXPosition;
    private float _spawnStartZPosition;
    private float _spawnEndZPosition;
    private float _finishXPosition;

    public override void Initialize()
    {
        _animator = GetComponent<Animator>();
    }
}
```

```

        // Определяем начало и конец по координате z для спавна
        агента
        float spawnPlatformZCenter =
        _spawnPlatform.transform.position.z;
        float spawnPlatformLength =
        _spawnPlatform.transform.localScale.z;
        _spawnStartZPosition = spawnPlatformZCenter -
        spawnPlatformLength / 2;
        _spawnEndZPosition = spawnPlatformZCenter +
        spawnPlatformLength / 2;

        _spawnXPosition = _spawnPlatform.transform.position.x;

        _finishXPosition =
        _finishPlatform.transform.position.x;
    }

    /* В начале эпизода устанавливаем случайный поворот,
    случайную позицию по координате z
    * и рассчитываем расстояние до финиша
    */
    public override void OnEpisodeBegin()
    {
        float spawnZPosition =
        Random.Range(_spawnStartZPosition, _spawnEndZPosition);

        transform.SetLocalPositionAndRotation(
            new Vector3(_spawnXPosition, _spawnYPosition,
            spawnZPosition),
            Quaternion.Euler(0f, Random.Range(-180f, 180f),
            0f));

        _lastDistance = _finishXPosition -
        transform.localPosition.x;
    }

    /* Добавление значений в входной вектор нейронной сети. Эти
    значения
    * добавляются к состоянию среды за которым наблюдает агент
    */
    public override void CollectObservations(VectorSensor
    sensor)
    {
        // Добавляем в наблюдение текущее значение расстояния
        до финиша
        sensor.AddObservation(_currentDistance);
        // Добавляем в наблюдение позицию агента - 3 координаты
        sensor.AddObservation(transform.localPosition);
        // Добавляем в наблюдение поворот агента - 4 компоненты
        Кватерниона
        sensor.AddObservation(transform.localRotation);
    }

```

```

public override void OnActionReceived(ActionBuffers actions)
{
    ActionSegment<int> discreteActions =
actions.DiscreteActions;

    // Ничего не делать
    if (discreteActions[0] == 0)
    {
        _animator.SetBool(_runAnimationParametr, false);
    }
    // Идти вперёд
    else if (discreteActions[0] == 1)
    {
        _animator.SetBool(_runAnimationParametr, true);

        transform.Translate(Vector3.forward * _moveSpeed *
Time.fixedDeltaTime);
    }
    // Идти назад
    else if (discreteActions[0] == 2)
    {
        _animator.SetBool(_runAnimationParametr, true);

        transform.Translate(Vector3.back * (_moveSpeed /
2f) * Time.fixedDeltaTime);
    }

    // Повернуться налево
    if (discreteActions[1] == 0)
    {
        transform.Rotate(new Vector3(0f, -1f, 0f) *
_turnSpeed);
    }
    // Повернуться направо
    else if (discreteActions[1] == 1)
    {
        transform.Rotate(new Vector3(0f, 1f, 0f) *
_turnSpeed);
    }

    // Рассчитываем текущее расстояние до финиша
    _currentDistance = _finishXPosition -
transform.localPosition.x;

    /* Сравниваем текущее расстояние до финиша с
расстоянием на предыдущем шаге.
    * Если он меньше добавляем награду, иначе уменьшаем.
Пересчитываем расстояние
    */
    if (_currentDistance < _lastDistance)
    {
        AddReward(0.3f);
        _lastDistance = _currentDistance;
    }
}

```

```

        }
        else
        {
            AddReward(-0.001f);
        }
    }

    /* Назначение действий агенту с помощью клавиатурного ввода.
    * Позволяет играть за агента
    */
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<int> actions =
        actionsOut.DiscreteActions;

        // При нажатии W назначить действие - идти вперёд
        if (Input.GetKey(KeyCode.W))
        {
            actions[0] = 1;
        }
        // При нажатии S назначить действие - идти назад
        if (Input.GetKey(KeyCode.S))
        {
            actions[0] = 2;
        }

        // Устанавливаем отсутствие поворота по-умолчанию
        actions[1] = 2;

        // При нажатии A назначить действие - повернуться
        // налево
        if (Input.GetKey(KeyCode.A))
        {
            actions[1] = 0;
        }
        // При нажатии D назначить действие - повернуться
        // направо
        if (Input.GetKey(KeyCode.D))
        {
            actions[1] = 1;
        }
    }

    private void OnTriggerEnter(Collider trigeeredCollider)
    {
        if (trigeeredCollider.gameObject.CompareTag(_carTag) ||
            trigeeredCollider.gameObject.CompareTag(_playerEndRoadTag) ||
            trigeeredCollider.gameObject.CompareTag(_bigNatureTag))
        {
            AddReward(-10f);

            EndEpisode();
        }
    }

```

```
        else if
(trigeeredCollider.gameObject.CompareTag(_finishPlatformTag)
)
    {
        AddReward(50f);

        EndEpisode();
    }
}
```

ПРИЛОЖЕНИЕ Б

(справочное)

Исследование стратегий в компьютерных играх с применением методов
искусственного интеллекта

Презентация

ИССЛЕДОВАНИЕ СТРАТЕГИЙ В КОМПЬЮТЕРНЫХ ИГРАХ С ПРИМЕНЕНИЕМ МЕТОДОВ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Автор работы: магистрант группы 22ВВИм2 Щегольков М.В.

Научный руководитель: д.т.н., профессор Зинкин С.А.

1

Цель, объект и предмет работы

- **Цель работы:** создание игры с внедрённым в неё ИИИ, рассмотрение и анализ различных алгоритмов ИИ и архитектур нейронной сети, формирование рекомендации для непосредственной реализации искусственного интеллекта в компьютерных играх.
- **Объект работы:** игровой искусственный интеллект.
- **Предмет работы:** глубокое обучение с подкреплением и компьютерные игры.

2

Игровой искусственный интеллект

Искусственный интеллект – система, которая имитирует процесс мышления человека. Имитация данного процесса может выражаться в решении определённых задач, принятии решений или выполнении творческих функций.

Игровой ИИ может проявляться как:

- то, что делает окружение реалистичнее, а графику красивее;
- то, что анализирует привычки и поведение игроков;
- то, что наделяет мышлением или его подобием неигровых персонажей (NPC).

3

Глубокое обучение

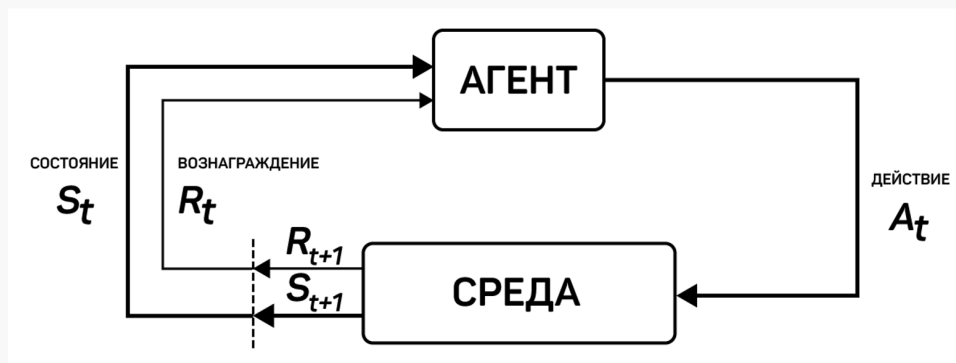
Обучение с подкреплением – метод, в котором используется корректировка стратегии по принципу проб и ошибок через обратную связь с данными, поступающими из окружающего мира

В обучении с подкреплением выделяют 2 основные группы алгоритмов:

- обучение без знания модели (Model-Free RL)
- обучение со знанием модели (Model-Based RL)

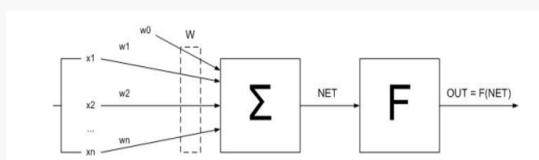
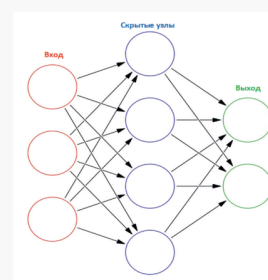
4

Взаимодействие между агентом и окружающей средой



5

Искусственный нейрон и нейронные сети



6

Основные алгоритмы обучения

- **Proximal Policy Optimization** – семейство методов первого порядка, которые стараются минимизировать функцию потерь при этом не сильно отходить от предыдущей политики.
- **Soft Actor-Critic** – в основе лежит наличие двух единиц – Актора, которые действует в среде, и Критика, который задаёт планку для Актора.

7

Стек технологий



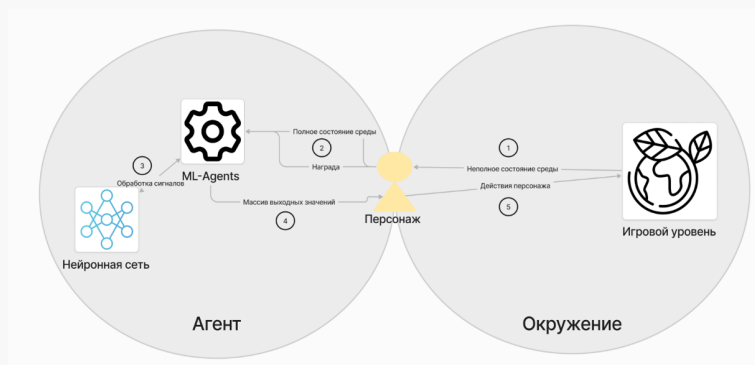
8

Окружение для обучения агента



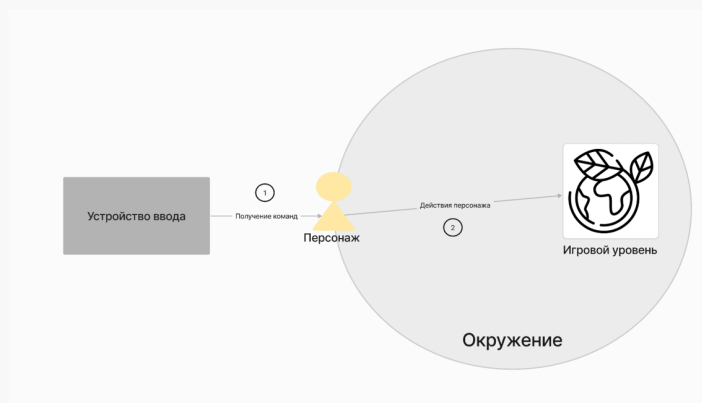
9

Взаимодействие компонентов в система с агентом



10

Схема управления персонажем с устройств ввода



11

Игра. Главное меню и режим против агента



12

Прохождение уровня агентами



13

Заключение

В ходе проведённой работы:

- Разработана игра, в которую внедрён игровой искусственный интеллект
- Описаны рекомендации для обучения агентов, разработан baseline для обучения агентов с помощью ML-Agents

Разработанная игра имеет большой потенциал для дальнейшего развития, которое может включать, например добавление новых режимов.

Полученные результаты исследования могут использоваться для дальнейших научных исследований в области обучения с подкреплением.

14

Спасибо за внимание!

15