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

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ИИТиМОИ
ФБОУ ВО «ЮУрГГПУ», к.п.н.

_____ Л.С. Носова

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка компьютерной игры с интеллектуальным поведением субъектов на Unity 3D

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ –09.04.04.2024.308-1491.ВКР

Научный руководитель,
доцент кафедры СП, к.п.н.
_____О.Н. Иванова

Автор работы,
студент группы КЭ-229
_____А.А. Кулаков

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

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

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-229
Кулакову Андрею Александровичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка компьютерной игры с интеллектуальным поведением субъектов на Unity 3D.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Бонд Д.Г. Unity и C#: Геймдев от идеи до реализации. // Питер, 2022. – № 2. – С.928.

3.2. Павлова А.И. Искусственные нейронные сети: учебное пособие. // Ай Пи Ар Медиа, 2021. – № 3. – С.190.

3.3. Lanham M. Learn Unity ML-Agents – Fundamentals of Unity Machine Learning // Packt Publishing Ltd, 2018. – №. 1. – P.204.

4. **Перечень подлежащих разработке вопросов**

4.1 Изучить предметную область.

4.2 Спроектировать игровое приложение.

4.3 Разработать алгоритм интеллектуального поведения субъектов в игре, использующий нейросетевые технологии.

4.4 Реализовать и протестировать компьютерную игру.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.п.н. О.Н. Иванова

Задание принял к исполнению А.А. Кулаков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В последние годы нейронные сети стали все более популярными в различных областях программного обеспечения. Они могут быть использованы для анализа и обработки данных, распознавания речи и естественного языка, компьютерного зрения и машинного обучения. Однако, развитие нейронных сетей не стоит на месте, и постоянно происходят новые открытия и инновации в этой области. Совершенствование алгоритмов и методов обучения, улучшение аппаратной базы и использование новых подходов позволяют создавать нейросети, которые могут решать задачи, недоступные ранее или улучшать те решения, которые уже были реализованы.

Современные игровые приложения становятся все более сложными и требуют от разработчиков использования новых технологий для улучшения игрового процесса и создания более реалистичного игрового мира [18]. Одной из таких технологий являются нейронные сети, возможности которых могут быть использованы в игровых приложениях для улучшения их функциональности.

Использование нейронных сетей в игровых приложениях может быть разнообразным. Они могут применяться для создания умных компьютерных противников, определения поведения персонажей в игре, управления камерой и освещением, распознавания голоса и жестов игроков, а также для создания реалистичной физики и графики.

Данная работа на тему использования нейронных сетей в игровых приложениях имеет большую актуальность в наше время. Она дает возможность изучить основные принципы работы нейронных сетей, их применение в игровых приложениях, возможности и позволяет проанализировать уже существующие решения. Кроме того, такая работа поможет выявить возможности для дальнейшего улучшения игровых приложений с помощью нейросетей.

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной игры с интеллектуальным поведением субъектов. Для достижения поставленной цели необходимо решить следующие задачи:

1) рассмотреть программные продукты и возможности движков для разработки игр;

2) рассмотреть инструмент Unity ML-Agents;

3) проанализировать публикации, в которых описано применение нейронных сетей и машинного обучения в игровых приложениях;

4) спроектировать игровой проект, который будет использовать нейронную сеть и машинное обучение.

Структура и содержание работы

Работа состоит из введения, двух глав, заключения и списка литературы. Объем работы составляет 74 страницы, объем списка литературы – 24 источника.

В первой главе описываются современные подходы в разработке игр, сравнительный анализ игровых движков и их инструментов для работы с машинным обучением, возможности модуля Unity ML-Agents, его особенности и недостатки, приводятся публикации игровых разработчиков.

Во второй главе описан процесс подготовки проекта, создания объектов окружения, создания игровых механик и обучение интеллектуального субъекта.

В третьей главе описано функциональное тестирование проекта.

В приложении А содержится код управления персонажем.

В приложении Б находится код поведения интеллектуального агента.

В приложении В находится код управления временем и событиями.

1. СОВРЕМЕННЫЕ ПОДХОДЫ В РАЗРАБОТКЕ ИГР

1.1. Описание предметной области

Нейронные сети – это математические модели, которые используются для анализа сложных данных и решения различных задач. В играх нейронные сети могут использоваться для создания умных компьютерных противников, анализа поведения игроков, оптимизации процесса генерации контента и многого другого.

Использование нейронных сетей в играх – это одна из самых актуальных тем в области компьютерных игр и искусственного интеллекта. Идея использования нейронных сетей для создания умных компьютерных противников в играх появилась еще в 90-х годах прошлого столетия. Однако, в те времена вычислительные мощности не были достаточными для реализации этой идеи на практике [17].

Сегодня же, благодаря развитию вычислительной техники, использование нейронных сетей в играх стало возможным [16]. Современные игры все чаще используют нейронные сети для создания более умных и адаптивных противников, улучшения графики и физики игрового мира.

Классические подходы к использованию нейронных сетей в играх включают следующее:

– создание более умных компьютерных противников, которые могут адаптироваться к действиям игрока и изменять свою стратегию;

– улучшение графики и физики игрового мира за счет реалистичного моделирования окружающей среды и ее взаимодействия с объектами в игре;

– оптимизация процесса генерации контента, такого как текстуры, звуковые эффекты, музыка и другие элементы игры;

– анализ данных поведения игрока и прогнозирование его действий.

Существует множество программных продуктов, которые используют нейронные сети для улучшения игрового опыта и используются игровыми движками для обеспечения качественного игрового процесса.

1. Nvidia DLSS – это революционная технология глубокого обучения, представленная компанией Nvidia, которая используется для повышения производительности и одновременно с этим качества графики в играх.

2. AMD FidelityFX Super Resolution – это аналог технологии DLSS, созданная компанией AMD для повышения производительности и качества графики в играх.

3. Unity ML-Agents – это платформа, которая предоставляет инструментарий для разработки игр с использованием машинного обучения и нейронных сетей на движке Unity.

4. DeepMind AlphaGo – это программа искусственного интеллекта, которая заняла первое место в игре «Го» против лучших игроков в мире.

5. TensorFlow – это библиотека от Google, которая используется для создания и обучения нейронных сетей. Она может быть использована для создания искусственного интеллекта в играх и системах обработки и анализа текстовых и графических данных [10].

6. PyTorch – это еще одна библиотека для машинного обучения, которая может быть использована для создания нейронных сетей для улучшения игрового опыта.

Также, есть российские компании, которые занимаются разработкой VR-технологий для игр, такие как «Fibrum» и «Deus Craft». Они используют нейронные сети для создания более реалистичных и интерактивных виртуальных миров.

Существует несколько игровых движков, которые могут использовать нейронные сети для различных целей.

1. Unity – один из наиболее популярных игровых движков, который имеет интеграцию с TensorFlow и Caffe2 фреймворками машинного обуче-

ния. Особенность Unity это простота его использования и поддержка NET-библиотек [15].

2. Unreal Engine – еще один очень популярный игровой движок, который поддерживает систему графического программирования «Blueprint» и может использовать нейронные сети для реализации различных функций, таких как генерация контента, управление ИИ и оптимизация производительности. Преимуществом пятой версии данного движка по сравнению с современными движками это использование таких технологий как Nanite, которые поддерживают рендеринг (визуализацию) высокополигональных моделей [6].

3. Godot Engine – бесплатный, с открытым исходным кодом игровой движок, который поддерживает Python и может использовать PyTorch для создания и обучения нейронных сетей. Главной составляющей данного движка является его доступность для каждого разработчика и его простота использования как для 2D, так и для 3D игр.

4. CryEngine – еще один мощный игровой движок, который может использовать нейронные сети для реализации различных функций, таких как управление ИИ и контроль анимаций персонажей [12]. Ключевая особенность данного движка заключается в его продвинутой системе освещения на сцене, что позволяет создавать атмосферную визуализацию.

5. Nau Engine – отечественный игровой движок, который был профинансирован компанией VK и на данный момент активно разрабатывается и проходит различные тестирования от разработчиков. Одной из главных особенностей данного ПО – поддержка современных технологий, такие как DLSS и «TensorFlow», которые были описаны ранее. Все особенности, которые анонсировали разработчики, позволят занять Nau Engine место среди лидеров зарубежных игровых движков.

Перечисленные движки имеют огромный спектр применения не только в играх, но и в создании фильмов, мультфильмов, в моделировании астрономических объектов, в образовании и медицине. Применение дан-

ных технологий позволяет упростить разработку и оптимизацию проектов, за счет чего данные инструменты обрели огромную популярность и являются стандартом во многих предприятиях [6].

1.2. Сравнительный анализ игровых движков

При выборе игрового движка для разработки проектов, включающих в себя задачи машинного обучения, важно учитывать особенности и инструменты каждой платформы. В данном сравнительном анализе рассмотрим преимущества и особенности в контексте работы с машинным обучением, разберем ключевые недостатки каждого движка:

1) Unity:

– широко используется в индустрии разработки игр;

– имеет интеграцию с различными библиотеками машинного обучения, такими как TensorFlow и PyTorch, через пакет Unity ML-Agents;

– некоторые функции могут требовать дополнительных платных плагинов;

2) Unreal Engine:

– мощные графические возможности с высоким качеством визуализации;

– имеет возможности для интеграции с библиотеками машинного обучения;

– более ресурсоемкий по сравнению с другими движками;

3) Godot Engine:

– бесплатный и с открытым исходным кодом;

– легкий для изучения и начала работы, имеет интуитивно понятный интерфейс;

– может иметь ограниченную поддержку библиотек машинного обучения по сравнению с Unity и Unreal Engine;

4) Nau Engine:

– ориентирован на разработку архитектурных и визуализационных проектов;

– может предоставлять хорошую производительность на проектах, не требующих сложного искусственного интеллекта;

– возможно более ограниченные возможности для работы с машинным обучением;

5) CryEngine:

– знаменит своими впечатляющими графическими возможностями;

– может предложить высококачественное визуальное представление проектов;

– может требовать больше времени настройки и оптимизации для работы с машинным обучением.

1.3. Виды интеллектуальных агентов

Существует множество различных типов агентов, которые используются в самых различных отраслях:

– чат-боты и виртуальные агенты, которые используются для автоматизированного взаимодействия с пользователем через текстовый или голосовой интерфейс;

– персональные ассистенты, помогающие пользователям в выполнении задач и получении информации, например: Siri от Apple, Google Assistant, Алиса от Яндекс, Amazon Alexa и Microsoft Cortana;

– автономные роботы или физические устройства, способные воспринимать окружающую среду и принимать решения на основе полученной информации, например: роботы-пылесосы, роботы-почтальоны, роботы-хирурги и т.д.;

– умный дом – один из типов агентов, которые могут помочь в управлении умными домашними системами, контроле энергопотребления,

автоматическом распознавании речи и звука, а также взаимодействии с пользователями для предоставления персонализированного опыта;

– экспертные системы, основанные на базе знаний экспертов в определенной области, например, может быть медицинская экспертная система, которая помогает диагностировать болезни и предоставлять лечебные рекомендации [2];

– агенты системы рекомендаций, которые используются для предоставления персонализированных рекомендаций пользователю на основе его предпочтений и поведения;

– агенты управления трафиком или интеллектуальные агенты, которые могут использоваться для оптимизации потока транспорта и управления трафиком на дорогах;

– в области финансов могут использоваться финансовые и торговые агенты, для прогнозирования цен на финансовых рынках, определения оптимальных стратегий и торговли, управления портфелем и рисками;

– интеллектуальные агенты могут помочь в области медицины, предоставляя поддержку при диагностировании болезней, анализируя медицинские данные и предоставляя лечебные рекомендации;

– интеллектуальные агенты в организациях, которые занимаются обработкой заявок, управление очередями и запасами, планированием производства и другие;

– агенты, способные играть в компьютерные игры или соревноваться с другими игроками.

1.4. Unity ML-Agents

Unity Machine Learning Agents (Unity ML-Agents) – это плагин, разработанный компанией Unity Technologies, который позволяет создавать и обучать агентов искусственного интеллекта (AI) для решения задач внутри игровых сред. Он предоставляет возможность разработчикам использовать

машинное обучение для создания интеллектуального поведения NPC (неконтролируемых персонажей) и других элементов игры.

Unity ML-Agents основан на библиотеке машинного обучения TensorFlow, что позволяет разработчикам использовать широкий спектр алгоритмов машинного обучения для тренировки агентов. Это включает в себя классические методы, такие как усиление обучения (reinforcement learning), а также современные алгоритмы глубокого обучения, такие как глубокие нейронные сети [3].

Одной из ключевых особенностей Unity ML-Agents является его интеграция с игровым движком Unity. Разработчики могут легко создавать среды для обучения агентов, используя инструменты Unity, такие как графический редактор сцен, физический движок и системы визуализации. Это позволяет им создавать сложные трехмерные среды, в которых агенты могут обучаться и развиваться. Unity ML-Agents поддерживает как симулированное обучение, где агенты тренируются внутри виртуальной среды, так и совместное обучение (cooperative-competitive training), где несколько агентов работают вместе или соревнуются друг с другом. Это открывает двери для создания разнообразных игровых сценариев, таких как командная игра, сражения и сотрудничество.

С помощью Unity ML-Agents разработчики могут создавать более умных и реалистичных персонажей в играх, а также использовать его для создания сценариев обучения с подкреплением и исследования AI. Данный плагин предоставляет инструменты и ресурсы, которые делают машинное обучение доступным для широкого круга разработчиков Unity, что способствует прогрессу в области разработки искусственного интеллекта в играх.

Каждый пользователь ML-Agents Toolkit может создать и использовать собственные алгоритмы для обучения. Это позволит контролировать поведение всех агентов на сцене с помощью Python. Также можно превратить свою среду в тренировочную площадку для обучения.

1.5. Обучение модели в Unity ML-Agents

ML-агенты предлагают два метода обучения с подкреплением.

1. Оптимизация проксимальной политики (PPO).
2. Мягкий актер-критик (SAC).

По умолчанию выбран алгоритм PPO, который является более универсальным и стабильным в сравнении с другими алгоритмами обучения с подкреплением.

SAC отличается от PPO тем, что он работает вне политики обучения, позволяя использовать накопленный опыт в прошлом. SAC использует буфер воспроизведения опыта, извлекая случайные сэмплы во время обучения. Это делает SAC более эффективным с точки зрения использования данных, зачастую требуя гораздо меньше образцов (в 5-10 раз меньше) для обучения на той же задаче, чем PPO. Однако SAC обычно требует большего числа обновлений модели. SAC рекомендуется для тяжелых или медленных сред (с шагом около 0.1 секунды или больше) и является алгоритмом «максимальной энтропии», позволяющим проводить внутренние исследования.

При обучении с подкреплением конечной целью агента является обнаружение поведения (политики), которое максимизирует вознаграждение. Для максимизации вознаграждения необходимо предоставить агенту один или несколько сигналов вознаграждения для использования во время обучения. Обычно награда определяется окружением и соответствует достижению какой-либо цели. Это то, что называется внешним вознаграждением, поскольку определяется вне алгоритма обучения.

Однако вознаграждения могут быть определены и вне окружающей среды, чтобы побудить агента вести себя определенным образом или помочь в изучении истинного внешнего вознаграждения. Эти вознаграждения называются внутренними сигналами вознаграждения. Общая награда, которую агент научится максимизировать, может представлять собой смесь внешних и внутренних сигналов вознаграждения.

Набор инструментов ML-Agents Toolkit позволяет определять сигналы вознаграждения модульным образом, которые можно смешивать и сопоставлять, чтобы помочь сформировать поведение агента и называются следующим образом:

– extrinsic: представляет вознаграждения, определенные в вашей среде, данный метод определен по умолчанию в Unity ML-Agents;

– gail: представляет собой внутренний сигнал вознаграждения;

– curiosity: представляет собой внутренний сигнал вознаграждения, который поощряет исследование в средах с редким вознаграждением;

– rnd: представляет собой внутренний сигнал вознаграждения, который поощряет исследование в средах с редким вознаграждением.

1.6. Особенности Unity ML-Agents

Технология Unity ML-Agents имеет ряд преимуществ и особенностей, которые выбирают как профессиональные разработчики, так и начинающие разработчики игр и различных симуляторов.

1. Unity ML-Agents обеспечивает удобство и быстроту настройки игры в качестве окружения для обучения нейронной сети. Разработчик сможет создавать интеллектуальных персонажей без наличия особых навыков программирования на языке C# или Python.

2. Инструментарий Unity ML-Agents предоставляет открытый исходный код, который можно свободно изменять и использовать в соответствии с лицензией «Apache 2.0». Это дает возможность модифицировать агентов и создавать собственные модели по усмотрению разработчика.

3. Unity ML-Agents содержит все необходимое для начала работы, включая готовые современные алгоритмы, подробную документацию и примеры проектов. Даже без опыта в области искусственного интеллекта или машинного обучения, разработчик сможет легко приступить к созданию и обучению интеллектуальных агентов. Кроме того, любой начинаю-

щий может обратиться к сообществу, где получит поддержку от других пользователей, либо от официальных представителей Unity.

4. С помощью Unity Inference Engine (Barracuda) есть возможность развертывать модели агентов на любой платформе, поддерживаемой Unity, включая ПК, мобильные устройства и консоли. Это обеспечивает гибкость и доступность для целевой аудитории.

5. Благодаря доступу к C#, протоколу связи и низкоуровневому API Python, разработчик получит возможность изменять или дорабатывать различные алгоритмы и методы обучения агентов, расширяя возможности улучшенного искусственного интеллекта, а любой unity-разработчик может исследовать различные варианты применения и настроить обучение агентов в соответствии с его потребностями.

6. Разработчик имеет возможность выбирать из разнообразных начальных окружений для создания 2D или 3D игр, систем непрерывного управления или крупных игровых пространств. Это даст большую свободу выбора при разработке и обучении агентов.

1.7. Недостатки Unity ML-Agents

Несмотря на выше описанные преимущества данного модуля Unity, у ML-Agents можно выявить ряд недостатков, которые характерны для любого готового инструмента для работы с нейронной сетью.

1. Ограниченная гибкость и расширяемость. Unity ML-Agents был разработан специально для игровой среды Unity, что может привести к ограничениям в его использовании для новых типов проектов или сред. Возможности расширения фреймворка за пределами игровой экосистемы Unity ограничены.

2. Высокие требования к вычислительным ресурсам. Обучение агентов с использованием глубокого обучения – это вычислительно интенсивный процесс, который потребует значительных вычислительных ресурсов, таких как графические процессоры (GPU) и высокопроизводительные

компьютеры. Если у разработчика нет доступа к достаточным вычислительным ресурсам, обучение агентов может быть замедлено или процесс вовсе может не состояться.

3. Необходимость большого объема данных для обучения. Чтобы достичь хороших результатов в обучении агентов, может потребоваться большой объем данных. Это означает, что вам может потребоваться провести множество эпох обучения и собрать достаточное количество взаимодействий с окружением, чтобы обучить агента на достаточно разнообразных входных данных.

Важно отметить, что некоторые из этих недостатков могут быть смягчены или преодолены с помощью дополнительной работы, настройки и опыта в использовании Unity ML-Agents.

1.8. Проекты, использующие Unity ML-Agents

С развитием игровой индустрии инструмент Unity ML-Agents постепенно обретает популярность у разработчиков [1]. Появляются новые жанры и проекты, которые выделяются своими новыми механикой, визуальным стилем, либо поведением персонажей.

В «Source of Madness» от студии Carry Castle, игроки погружаются в увлекательное приключение, где каждое прохождение привносит новые вызовы благодаря постоянно меняющемуся и динамичному миру (рисунок 1). В этом захватывающем экшене игроки сталкиваются с огромным разнообразием процедурно генерируемых монстров, которые заставят их использовать свои навыки и тактику.

Команда разработчиков, работая над созданием этих запоминающихся монстров, столкнулась с несколькими интересными задачами. Одной из них была особая физика, которая лежит в основе управления монстрами. Эта физика является необычной и требовала тщательной работы для создания реалистичного и захватывающего игрового опыта.

Рисунок 1 – Обложка игры Source of Madness

Другой важной задачей было создание огромного количества различных видов монстров (рисунок 2). Команда разработчиков стремилась достичь максимального разнообразия, чтобы каждый встречаемый монстр был уникален и представлял свои угрозы и слабости. Такое разнообразие добавляет глубину и повышает заинтересованность игроков, которые всегда будут ожидать новых вызовов.

Рисунок 2 – Сгенерированные монстры в игре Source of Madness

Чтобы обеспечить монстрам естественное и реалистичное поведение, команда разработчиков использовала инструментарий ML-Agents, специализирующийся на глубоком обучении с подкреплением. С помощью этого инструментария они обучили модель нейронной сети, которая позволяет монстрам демонстрировать умные и адаптивные действия во время игровых ситуаций.

Данный проект был одним из самых первых, который использовал игровых интеллектуальных агентов, данная игра доступна на множестве игровых платформ и продается в игровых магазинах «Steam», «GOG» и «Microsoft».

Игра «Puppo The Corgi» – 3D кроссплатформенная игра, разработанная на Unity, в которой игрок бросает палку, а собака по имени Пуппо должен её принести обратно (рисунок 3). Разработчик данной игры написал статью на официальном сайте «unity.com», где описывает процесс создания данного проекта, сопровождая все медиа-ресурсами. Отличительной особенностью этой мини-игры является то, что движения собаки не задаются анимацией или скриптами, а обучаются с помощью методов обучения с подкреплением. Движения песика были обучены с использованием усиленного обучения, а его поведение определяется физическим движком.

Рисунок 3 – Обложка игры «Puppo The Corgi»

Для обучения Пуппо был использован алгоритм обучения с подкреплением, схожий с обучением щенка приносить палку. Сначала Пуппо блу-

ждает вокруг и не знает, что делать, но после каждого успешного выполнения задачи (подбора палки) получает вознаграждение. Постепенно он узнает, что подбирание палки является способом получить вознаграждение, и продолжает выполнять эту задачу (рисунок 4).

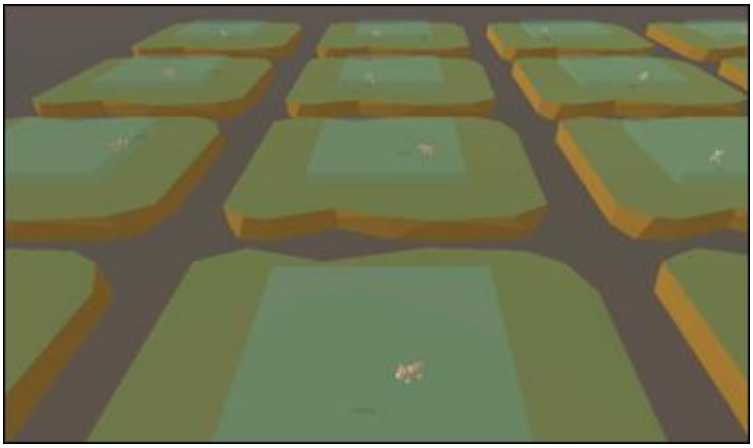

Рисунок 4 – Процесс обучения собаки Пуппо

Результаты обучения удивили разработчиков: в течение двух часов обучения Пуппо научился эффективно бегать к цели. К тому времени, как обучение продолжалось, он освоил навыки бега и даже научился вставать, если переворачивается. Это неуклюжее поведение выглядит очень мило и может быть использовано в игре. Помимо этого в игре собираются наблюдения о состоянии окружающей среды (близость к цели, положение ног и т.д.), чтобы принять решение о следующем действии, а затем получает вознаграждение в зависимости от выполненного действия. За счет физического движка Unity и его возможностям, собака в игре взаимодействует с окружающим миром и объектами, которые расположены на карте, что делает задачи и их решения разнообразными.

Игра получила популярность на выставке Unite Berlin и распространяется бесплатно.

Игра «Kart Racing» – популярный шаблон гоночных 3D-игр, где игроки имеют возможность управлять небольшими машинами (рисунок 5).

Она служит основой для современных игр, использующих передовые технологии, такие как нейронные сети и интеллектуальные агенты.

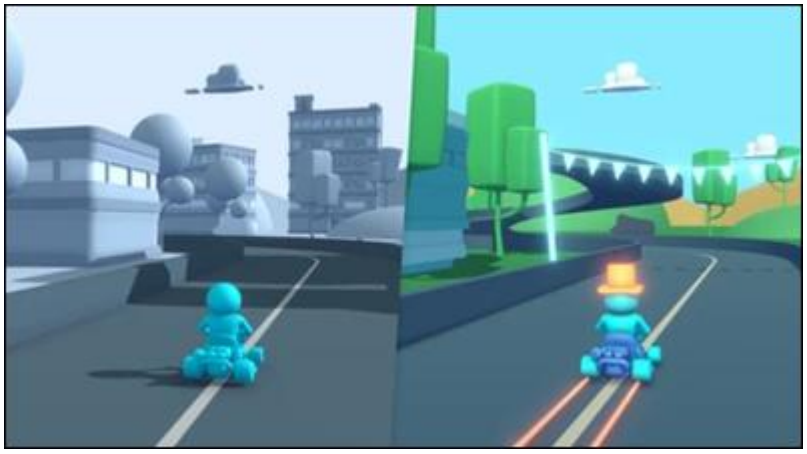

Рисунок 5 – Главное изображение игры Kart Racing

Эта микроигра разработана на Unity ML-Agents и доступна бесплатно на официальном сайте Unity. Её использование в качестве основы для различных проектов стало возможным благодаря функциональности Unity ML-Agents, что позволяет создавать реалистичное поведение компьютерных противников и обучать их с помощью искусственного интеллекта.

Данная игра также получила большую популярность среди сторонних разработчиков, которые создали множество любительских модификаций (рисунок 6). Это открывает возможности для создания собственных трасс и гоночных сценариев, а также добавления продвинутых систем искусственного интеллекта. Благодаря этому, игроки не только могут разрабатывать свои собственные гонки с продвинутым искусственным интеллектом, но и наслаждаться модификацией игры, созданной другими энтузиастами.

Проект «Kart Racing» позволяет разработчикам игр вдохновиться и подтолкнуть к изучению и внедрению в свои проекты новой технологии Unity ML-Agents.

Малоизвестный проект «AI Walker Albert» – это увлекательная демонстрационная игра, разработанная с использованием Unity ML-Agents.

Суть этой демонстрационной игры заключается в наблюдении за процессом обучения продвинутого искусственного интеллекта в ходьбе. Процесс обучения происходит поэтапно, позволяя агенту осваивать различные навыки, необходимые для успешной ходьбы и преодоления препятствий.

Рисунок 6 – Одна из модификаций игры

В начале данной игры искусственный интеллект должен ползти по ровному ландшафту, чтобы достигнуть заданной точки за определенное время. Это помогает ему осознать окружающую среду и научиться перемещаться в пространстве (рисунок 7).

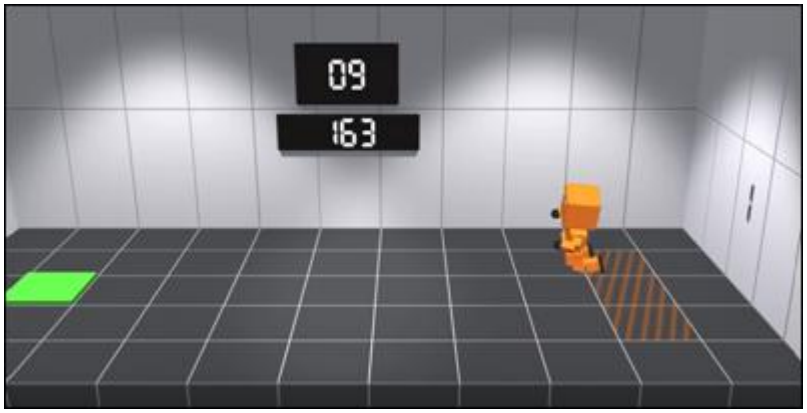

Рисунок 7 – Попытки ИИ дойти до точки финиша

Затем агенту предстоит встать на ноги и сделать первые шаги, стараясь достичь целевой точки без падений. Начальные попытки могут быть

несколько неуклюжими, но с каждой попыткой интеллектуальный агент становится все лучше и лучше, улучшая свои навыки ходьбы (рисунок 8).

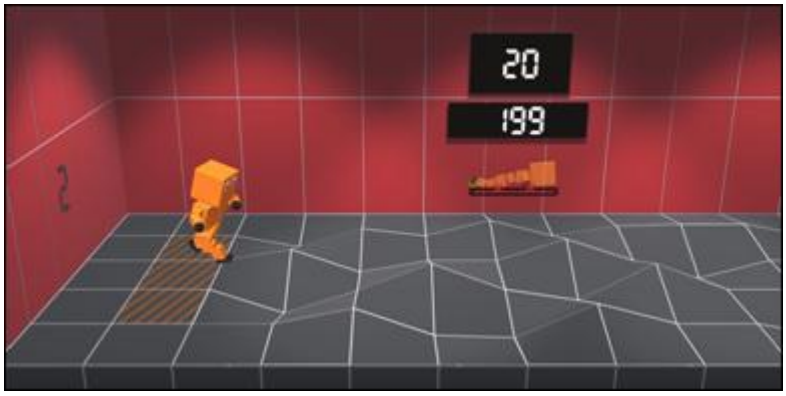

Рисунок 8 – Агент пытается ходить

Когда агент достигает стабильности в ходьбе, ему предоставляются более сложные задания для достижения конечной точки, чтобы он держался на ногах максимально уверенно: в него кидают предметы, ставят преграды в виде стен, делают изогнутый ландшафт и прочее, что может поспособствовать его улучшению навыков перемещения на ногах (рисунок 9).

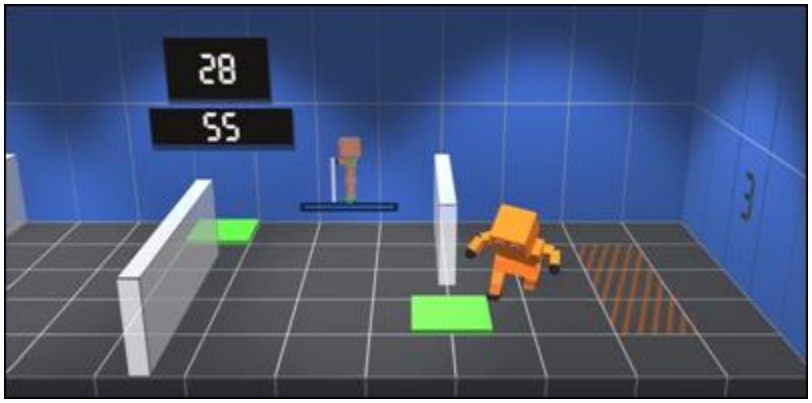

Рисунок 9 – Агенту необходимо преодолеть препятствия

В финальном задании агенту предстоит пройти длинный уровень, полный различных препятствий и помех. Его цель – завершить уровень, не теряя равновесие и не падая (рисунок 10).

«AI Walker Albert» восхищает своими возможностями и продемонстрирует, как через поэтапное обучение искусственный интеллект может

достичь мастерства в ходьбе. Эта игра вдохновляет разработчиков и исследователей на изучение применения Unity ML-Agents и развитие новых техник обучения для создания интеллектуальных агентов в игровом мире.

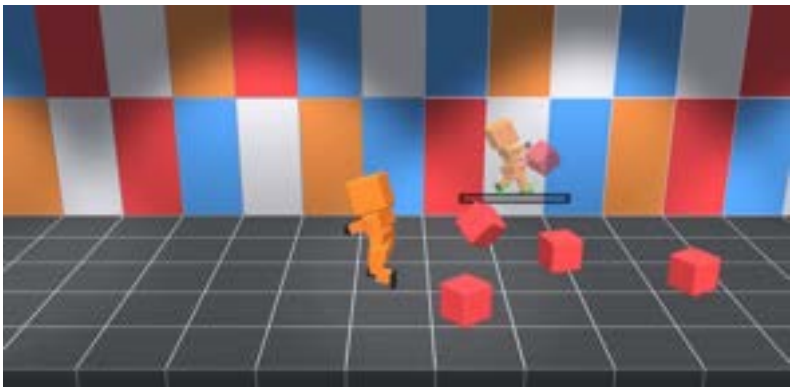

Рисунок 10 – Агент «проходит» финальное задание

1.9. Статьи и исследования разработчиков

В научных Интернет-ресурсах, например, на «elibrary.ru» и «researchgate.net» можно ознакомиться с публикациями различных авторов, которые демонстрируют свои наработки или делятся своими исследованиями в области изучения нейронных сетей и машинного обучения на движке Unity.

В статье Сокова И.А. «Влияние изменения параметров среды обучения для нейронных сетей в ML-Agents Unity» [11] рассматривается обучение нейронной сети с изменением среды обучения при использовании пакета ML-Agents. Автор исследует возможность изменения среды обучения и параметров «наград» агента для достижения желаемых результатов обучения. Для этого была использована методика постепенного до обучения. Исследование проводилось на платформе Unity с использованием агента, который должен был точно поражать мишени на платформе. Обучение проходило в несколько этапов, где среда обучения и параметры награды постепенно изменялись. Нейронная сеть состояла из 256 нейронов и тренировалась с помощью метода PPO. Всего одновременно тренировалось 32 агента.

На первом этапе обучения агент стрелял лучами во все стороны, чтобы быстро поразить мишени. На втором этапе появились препятствия в виде стен, и агенты научились быстро перемещаться между ними. На третьем этапе была добавлена отрицательная награда за попадание в стены, и агенты перестали стрелять в стены. На последнем этапе параметры награды вернулись к начальным значениям, и количество мишеней было увеличено до 5.

После окончания обучения агенты успешно проходили всю область платформы и поражали только мишени. Таким образом, автор пришел к выводу, что постепенное обучение нейронных сетей с изменением параметров среды является эффективным подходом, который можно реализовать с помощью ML Agents в Unity.

В заключении отмечается, что одним из недостатков данного подхода является необходимость остановки и перезапуска процесса обучения из командной строки. Однако благодаря возможности автоматического сохранения промежуточных значений весов нейронной сети, обучение может продолжаться продолжительное время. Отмечается, что пакет ML Agents Unity предоставляет разработчикам эффективный способ обучения нейронных сетей.

В статье Козлова Д.А. и Мясникова В.В. под названием «Влияние состава наблюдений окружающей среды в задаче приобретения навыков передвижения в трёхмерном пространстве при использовании алгоритмов обучения с подкреплением» [4] рассматривается задача обучения передвижению в трехмерном пространстве для человекоподобных мехатронных устройств. Для решения этой задачи используется метод обучения с подкреплением (RL). Основным алгоритмом RL, рассмотренным в работе, является Soft Actor Critic (SAC), который показал хорошую эффективность в решении подобных задач.

В исследовании исследуется влияние объема и состава наблюдений об окружающей среде на эффективность решения задачи с использованием

RL-метода [5]. Для экспериментов использовался игровой движок Unity вместе с пакетом ML-Agents. Была создана модель двуногого автономного устройства с двумя подвижными конечностями. Задачей агента было достичь предопределенного места в 3D среде. Различные виды наблюдений были переданы агенту в каждом эксперименте. В результате экспериментов выяснилось, что информация о положении и угле суставов, а также о приложенной к ним силе, оказывает отрицательное влияние на качество решения задачи, несмотря на их полезность для обучения.

Результаты эксперимента показали, что информация о положении, угле суставов и силе в суставах является избыточной и отрицательно влияет на качество решения задачи обучения передвижению. Лучший результат был достигнут при использовании ограниченного набора наблюдений [14].

В заключении авторы подводят итоги эксперимента и обобщают его результаты.

Помимо публикаций сторонних разработчиков и исследователей, разработчики из компании Unity ML-Agents предоставляют свои мануалы и наблюдения касательно данного инструмента, а также делятся своими мнениями.

В статье «Unity: A General Platform for Intelligent Agents» [22], которая была опубликована официальными представителями Unity Technologies: Дени Лэнгом, Венсан-Пьером Берже и Артуром Джулиани, рассказывается о новом инструментарии для создания и взаимодействия с симуляционными средами с использованием платформы Unity. Разработчики предлагают открытый исходный код и набор инструментов под названием Unity ML-Agents Toolkit.

Статья начинается с упоминания о том, что современные достижения в области глубокого обучения с подкреплением и робототехники обусловлены наличием все более реалистичных и сложных симуляционных сред. Однако многие из существующих платформ предоставляют нереалистичную визуализацию, неточную физику, низкую сложность задач или огра-

ниченную возможность взаимодействия между искусственными агентами. Кроме того, многие платформы не имеют возможности гибко настраивать симуляцию, что делает ее черным ящиком с точки зрения системы обучения.

Затем авторы описывают Unity ML-Agents Toolkit и его преимущества. Они отмечают, что благодаря использованию Unity в качестве платформы для симуляции, инструментарий позволяет создавать среды обучения, которые богаты сенсорной и физической сложностью, предлагают увлекательные когнитивные задачи и поддерживают динамическое взаимодействие между множеством искусственных агентов. Описывается дизайн платформы, протокол коммуникации, набор примеров сред и различные сценарии обучения.

В следующем разделе статьи рассматриваются другие платформы симуляции, такие как Arcade Learning Environment (ALE), DeepMind Lab, Project Malmo, Mujoco и VizDoom. Для каждой платформы представлено краткое описание и указана ее роль в развитии более продвинутых алгоритмов и моделей глубокого обучения с подкреплением.

Затем авторы переходят к описанию платформы Unity и ее особенностей. Unity – это платформа для разработки игр, включающая игровой движок и графический пользовательский интерфейс. В статье приводится терминология, связанная с Unity, и описывается его функциональность, основные свойства и возможности создания сред с высокой степенью сложности визуальных, физических, когнитивных и социальных аспектов.

Далее рассматривается предыдущее использование Unity в качестве симуляции, такое как AI2Thor и платформа, разработанная Cornell, а также примеры успешного использования возможностей Unity для обучения моделей на симуляциях и последующего переноса их на реальные робототехнические системы.

Теперь проанализируем статьи с сайтов, где разработчики делятся своими достижениями с другими людьми, которые вдохновляют начинающих разработчиков игр на движке Unity и не только.

Статья, написанная авторами Деррилом Чарльзом и Стивеном Макглинчи «Прошлое, настоящее и будущее искусственных нейронных сетей в цифровых играх» («The past, present and future artificial neural networks in digital games») [19] описывает начало применения нейронных сетей в играх в 90-х годах и их развитие в настоящее время [21].

Авторы статьи рассуждают и приводят ряд примеров использования нейронных сетей в играх, например:

– для создания более умных и реалистичных врагов или союзников в играх;

– для генерации уровней, персонажей, предметов и другого контента, что может увеличить интерес и продолжительность игры [7];

– в качестве анализа поведения игроков и предсказания их действий на основе имеющихся данных [7];

– в обработке естественного языка в играх, что может помочь создавать более реалистичные и глубокие диалоги между персонажами или фильтровать игровые чаты;

– для управления поведением персонажей в играх, влиять на их анимации и внешний вид, что может привести к более динамичной и разнообразной игре;

– для оптимизации производительности игры, например, путем автоматической настройки параметров графики или установки приоритетов для различных задач.

Данные авторы статьи также обсуждают перспективы будущего применения нейронных сетей. Они упоминают технологии, которые используются для создания более сложных и реалистичных нейронных сетей, таких как генеративные модели и усиленное обучение.

Кроме того, статья описывает преимущества и недостатки использования нейронных сетей в играх. Например, авторы указывают на то, что нейронные сети могут значительно повысить сложность и реалистичность игры, но это может повлечь за собой значительные затраты на вычислительные ресурсы и увеличение времени разработки игровых приложений.

В целом, статья демонстрирует значительный потенциал применения нейронных сетей в игровой индустрии и предлагает возможности для будущих исследований в этой области.

В статье, которая была написана авторами сайта «Plato AI Stream» [7] в 2023 году, рассказывается о главных преимуществах разработки игр с использованием ИИ (искусственного интеллекта) и машинного обучения, помимо этого авторы выделяют несколько основных преимуществ такого подхода.

1. Персонализированный пользовательский опыт. Использование искусственного интеллекта в играх позволяет создавать персонализированные видеоигры, учитывая предпочтения каждого игрока благодаря анализу их игровых привычек. Это позволяет рекомендовать игровым компаниям индивидуальные игровые возможности, контент, внутриигровые задачи и награды для каждого игрока [12].

2. Продвинутые игровые персонажи. Применение методов машинного обучения и ИИ позволяет делать этих персонажей более гиперреалистичными и адаптивными к действиям игрока при помощи алгоритмов усиления обучения.

3. Непредсказуемость. Использование машинного обучения и искусственного интеллекта в разработке игр позволяет избежать предсказуемости игрового дизайна и поддерживать интересность и свежесть игры на протяжении долгого времени. Такой подход делает невозможным гарантировать, что произойдет в игре дальше, и способствует созданию недетерминированных сюжетных линий, которые увеличивают долгосрочную перспективу игры [13].

4. Игровая аналитика и предиктивная аналитика. Чтобы помочь разработчикам лучше понимать своих пользователей и создавать более привлекательные игры необходимо анализировать данные пользователей [10]. Использование методов и техник машинного обучения и искусственного интеллекта помогает автоматизировать процесс анализа больших объемов данных, что позволяет быстро получать ценную информацию о поведении игроков и эффективности игровых механик.

Помимо этого, в статье приводятся примеры игр, которые используют машинное обучение и разрабатывались с поддержкой нейронных сетей, такие как: PUBG, FIFA, GTA5, Minecraft, Clash of Clans, Fortnite и Overwatch. Данные игры использовали такие системы как анализ поведения игроков, обучение с подкреплением, стабилизацию игровой сложности и улучшение качества визуализации, за счет чего игры обрели огромную популярность на мировом рынке.

В заключении данной статьи авторы уверяют, что искусственный интеллект и машинное обучение играют важную роль в разработке игр, создавая более захватывающий игровой процесс. Они изменяют все аспекты игр, начиная от поведения неигровых персонажей и заканчивая процедурной генерацией, игровой аналитикой и обработкой естественного языка [12]. Эти технологии революционизируют процесс разработки игр. Автор подчеркивает, что будущее обещает еще более передовые приложения и возможности в игровой составляющей.

Статья, написанная автором Алессией Нигретти «Использование агентов машинного обучения в настоящей игре: начинающий гид» (Using Machine Learning Agents Toolkit in a real game) [24] в официальном блоге разработчиков Unity, посвящена использованию агентов машинного обучения (Machine Learning Agents, сокращенно ML-Agents) в разработке игры на платформе Unity [24].

Данная публикация представляет собой руководство для начинающих, которые хотят создать свою игру с использованием технологии ма-

шинного обучения. Автор статьи объясняет, что такое ML-Agents и как они работают в Unity, а также приводит примеры использования этой технологии в играх (рисунок 11).

Рисунок 11 – Одна из предустановленных моделей ML-Agents

ML-Agents – это библиотека, разработанная Unity Technologies, которая позволяет разработчикам создавать агентов (персонажей с искусственным интеллектом) с помощью машинного обучения. Эти агенты могут обучаться взаимодействовать с окружением, принимать решения и выполнять определенные действия [20]. Алессия Нигретти показывает на практике каким образом можно создавать и обучать модель на готовом игровом движке, который предоставляет своим разработчикам все необходимые для этого библиотеки.

Далее автор статьи объясняет, как создать своего первого агента машинного обучения в Unity и как настроить его поведение, чтобы он мог учиться решать задачи в игре. Он также даёт несколько советов и рекомендаций по поводу подбора параметров и настройки агента, чтобы он достигал лучших результатов.

В конце статьи автор дает несколько полезных советов и рекомендаций по использованию ML-Agents в реальной игре, а также ссылки на другие ресурсы, которые помогут начинающим разработчикам продвинуться в изучении машинного обучения и создании своих игр. В целом, статья является полезным руководством для начинающих разработчиков, которые

хотят использовать технологию машинного обучения в своей игре на платформе Unity [15].

Статья на сайте «gamedev.net» под названием «Unity ML-Agents» посвящена установке и демонстрации возможностей инструмента ML-Agents для разработки и обучения искусственных нейронных сетей в игровых приложениях [23].

Майкл Лэнхэм, автор данной статьи подробно описывает процесс установки и настройки всех необходимых модулей для разработки приложений с использованием нейронных сетей, такие как Unity ML-Agents и «TensorFlow». После установки необходимых модулей, в данной публикации автор визуализирует процесс обучения своей готовой модели с помощью инструмента «TensorBoard» (рисунок 12), который входит в пакет «TensorFlow», установленный ранее.

Рисунок 12 – Вывод графиков с помощью инструмента «TensorBoard»

Помимо этого, в публикации разбирается распаковка и открытие предустановленных обученных моделей от компании Unity, которые охватывают множество типов различных проектов, а также все необходимые составляющие для создания и обучения своих моделей.

Публикация от компании «Pixonic», опубликованная на сайте «Habr.com» под названием «Машинное обучение в Unity: учим МО-агентов перепрыгивать через стены» [8] рассматривает конкретный пример работы о том, как использовать машинное обучение для создания искусст-

венного интеллекта, способного перепрыгивать через стены в игровом движке Unity.

Автор статьи начинает с объяснения основных понятий и принципов машинного обучения, например, обучения с подкреплением (reinforcement learning), глубокого обучения (deep learning) и других [9]. Он также описывает библиотеку «TensorFlow» и ее возможности для создания моделей машинного обучения [23].

Затем автор переходит к созданию простой игровой среды в Unity и настройке ее для обучения агентов (рисунок 13). В данном случае среда представляет собой поверхность с препятствиями (стенами), которые агент должен перепрыгнуть, чтобы достичь цели [8].

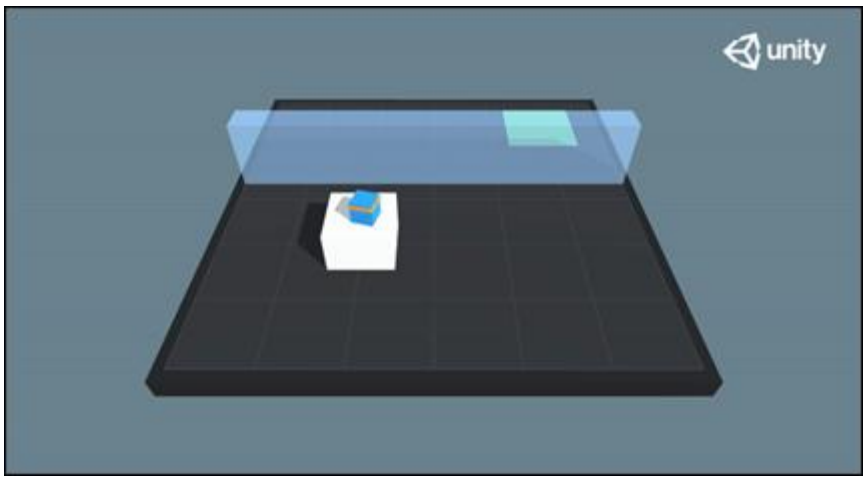

Рисунок 13 – Демонстрация работы программы

Далее автор пошагово описывает процесс обучения агента с помощью библиотеки «TensorFlow» [10]. Он объясняет, какие данные необходимо собрать для обучения агента, как определить функцию награды (reward function), как выбрать алгоритм обучения и как настроить параметры модели (рисунок 14).

В конце статьи автор представляет свой собственный алгоритм обучения, основанный на глубоком обучении, который показал лучшие результаты, чем традиционные методы машинного обучения.

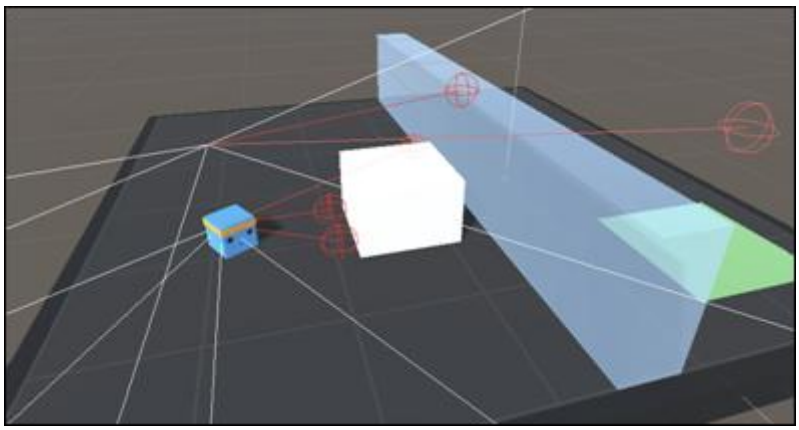

Рисунок 14 – Визуализация рейкастов и целей

Вывод по первому разделу

В данной главе мы рассмотрели инструменты и технологии, которые используют как начинающие разработчики, так и профессиональные студии по разработке игр. Узнали, что каждый из современных игровых движков имеет свои сильные и уникальные особенности, которые помогут при разработке определенных проектов. Рассмотрели технологии машинного обучения и нейросети, которые могут задействоваться в проекте, и работа которых не только улучшит игровой процесс, но и обеспечит качество игрового продукта.

Были изучены пять различных публикаций, авторы которых описывают, как использовались нейронные сети и машинное обучение в конце прошлого столетия, объясняют причину использования этих технологий именно сейчас и рассказывают, как современные технологии позволяют реализовать нейронные сети не только в профессиональном программном обеспечении, но и в играх. Так же были рассмотрены статьи по практическому применению машинного обучения в играх на движке Unity с помощью модуля ML-Agents, настройка этой утилиты на игровой платформе.

2. ПРОЕКТИРОВАНИЕ ИГРЫ НА ИГРОВОМ ДВИЖКЕ UNITY

2.1. Подготовка проекта и подключение модуля ML-Agents

Перед тем, как начать работу с Unity Ml-Agents, нужно пройти ряд действий для подготовки к созданию проекта. Данными шагами является установка необходимого программного обеспечения для создания нашего проекта.

План работы по установке Python

1. Первый шаг – при установке Python необходимо перейти на официальный сайт Python (https://www.python.org) и скачать версию, которая совместима с последней версией Unity ML-Agents, данная версия питона позволит корректно работать с данным инструментом.

2. Следующий шаг – запуск установочного файла и следование инструкциям мастера установки. Необходимо убедиться, что опция «Add Python to PATH» выбрана для удобства использования Python из командной строки.

3. Последний шаг – следует убедиться и проверить завершенную установку Python, это можно реализовать путем проверки версии нашей установленной программы. Для этого необходимо ввести необходимую команду в командную строку Windows (рисунок 15).

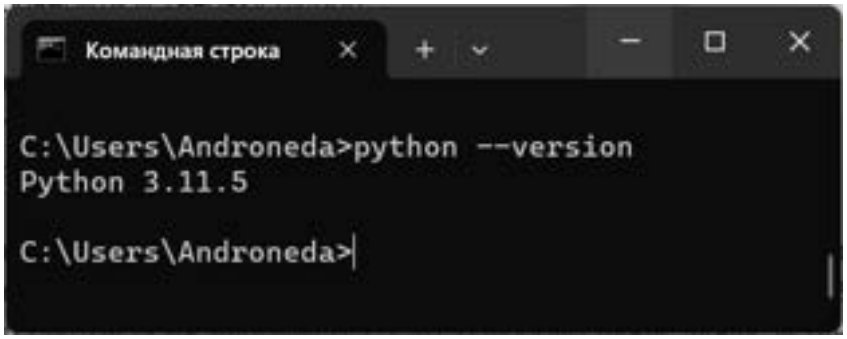

Рисунок 15 – Проверка установленной версии Python

Установка фреймворка «TensorFlow»

Установка «TensorFlow» требует активной установки Python, поэтому необходимо убедиться, что Python успешно установлен на предыдущем

шаге. Откроем командную строку (терминал) и выполним следующую команду для установки «TensorFlow» с использованием команды «pip», менеджера пакетов Python (рисунок 16).

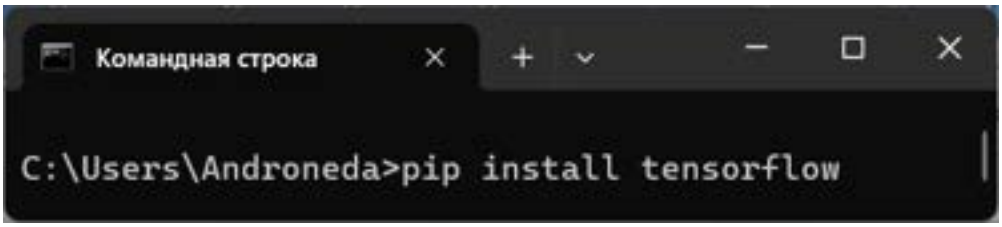

Рисунок 16 – Команда для установки актуальной версии «TensorFlow»

Установка фреймворка «PyTorch»

Чтобы установить фреймворк «PyTorch», необходимо зайти на официальный сайт данного ПО и выбрать необходимые настройки для генерации консольной команды (рисунок 17).

Рисунок 17 – Проверка установленной версии «TensorFlow»

Установка игрового движка Unity

1. ML-Agents является расширением Unity, поэтому для его установки потребуется наличие среды разработки Unity. Для этого необходимо перейти на официальный сайт Unity (https://unity3d.com) и скачать последнюю версию «Unity Hub» в соответствии с операционной системой пользователя.

2. Во время установки Unity Hub и необходимо следовать инструкциям мастера установки и зарегистрировать пользователя (если учетная запись ещё не создана).

3. После установки Unity Hub нужно открыть его и перейти на вкладку «Installs», щелкнуть по кнопке «Add» и выбрать необходимую версию Unity для установки, после этого нажмите на кнопку «Install». Для нашего проекта, при установке движка, не понадобится никаких дополнительных компонентов, поэтому можно пропустить выбор дополнительных надстроек при инсталляции Unity.

4. После успешного завершения мастер установки покажет соответствующее уведомление и последующие рекомендации.

Скачивание и подключение пакета ML-Agents к Unity

1. Для работы с пакетом инструментов ML-Agents в Unity, сначала нужно скачать этот пакет с официального сайта (https://unity-technologies.github.io/ml-agents/), либо можно воспользоваться консольной командой и скачать его через командную строку, как показано на рисунке 18.

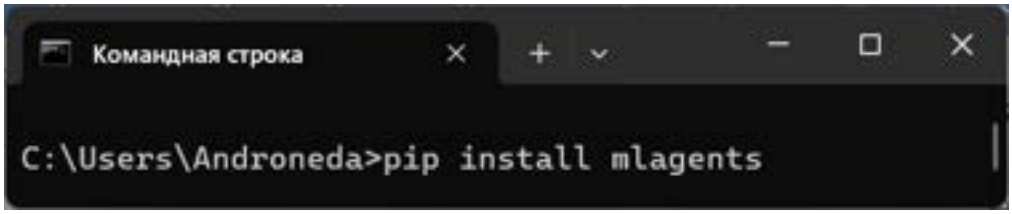

Рисунок 18 –Установка пакета ML-Agents через командную строку

2. После этого необходимо создать новый проект, в который будет импортирован данный инструмент машинного обучения. Для этого нужно запустить менеджер проектов Unity Hub, во вкладке «Projects» нажать на кнопку «New project». После этого выберем нужную установленную версию Unity (используем последнюю актуальную версию для корректной работы ML-Agents), выберем тип проекта «3D», которое означает, что проект будет иметь 3D-сцены и все необходимее инструменты, для проектирова-

ния такого типа проекта укажем название «Porter» и расположение проекта как показано на рисунке 19.

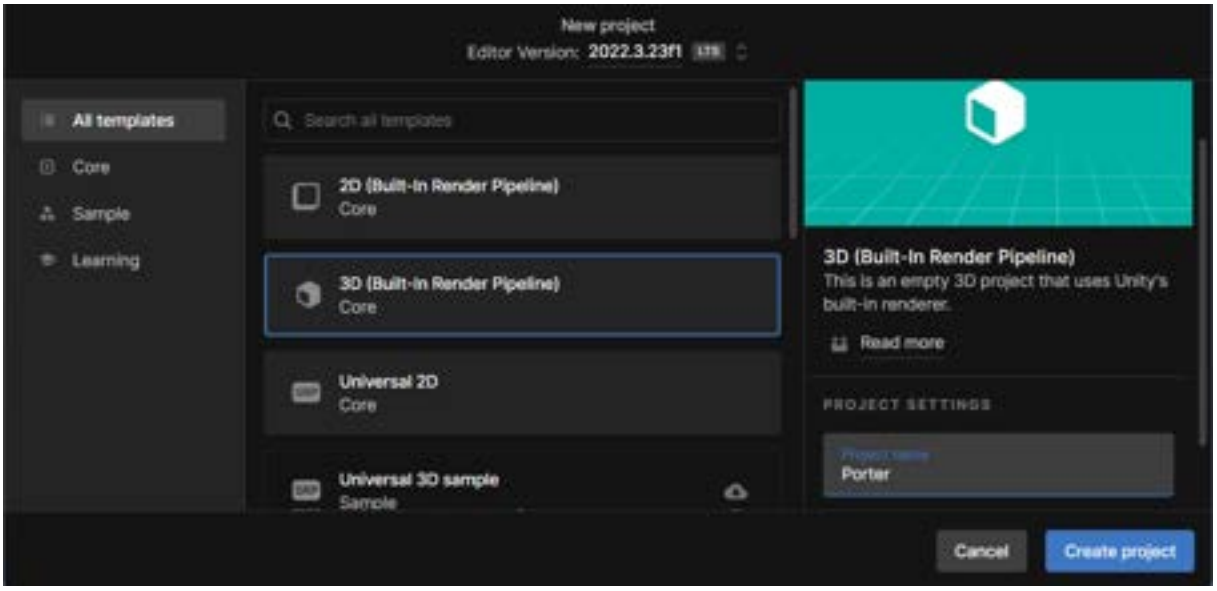

Рисунок 19 – Окно создания проекта Unity

3. Теперь созданный проект Unity готов к проектированию игры, в него можно импортировать пакет ML-Agents. Для этого надо в верхнем меню перейти в группу «Assets», выбрать в списке «Package Manager». Затем появится окно импорта пакетов (рисунок 20).

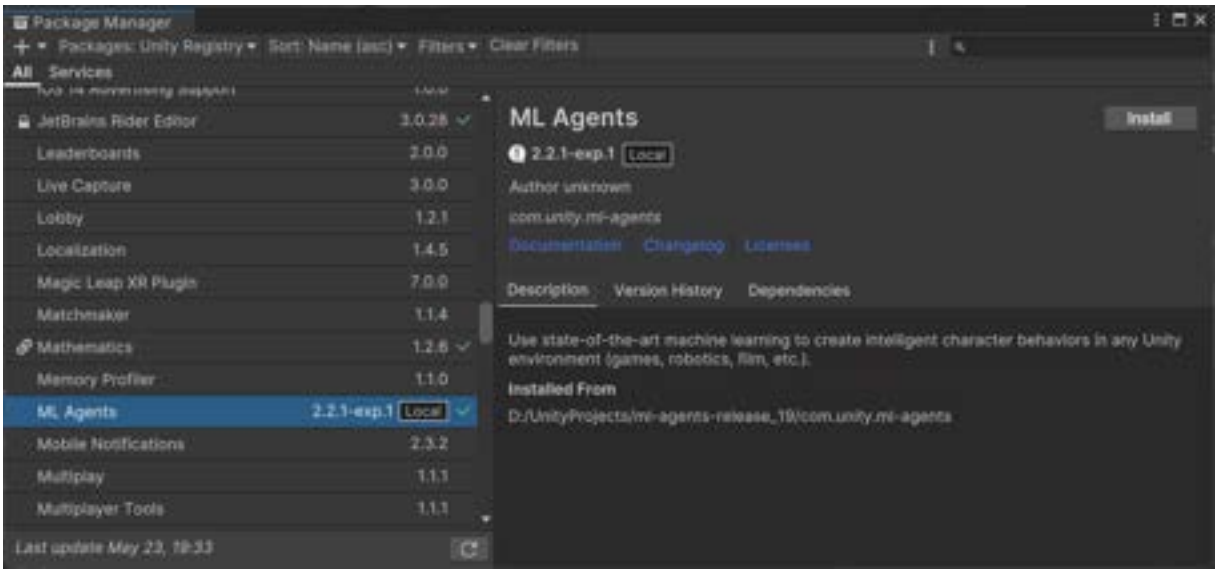

Рисунок 20 – Подключение ML-Agents к созданному проекту

4. Выберем в левом списке «Unity Registry», затем в среднем списке «ML Agents», после этого представятся подробности об этом модуле и появится возможность установить его в данный проект, нажав на кнопку «Install».

После установки и импорта всех необходимых компонентов, данный проект имеет все необходимые программные модули, теперь можно приступать к размещению готовых моделей и игровых элементов в игровой местности.

2.2. Проектирование трехмерных элементов окружения

Трехмерные элементы окружения – это объекты, которые будут располагаться на сцене (карте). В данном случае наши объекты будут располагаться на открытой игровой трехмерной местности, поэтому будем создавать декоративные объекты (дома, ограждения и столбы) и текстуры для них.

Рассмотрим инструменты, которые понадобятся для поставленной задачи.

1. Blender – это бесплатная и открытая программа для создания трехмерной графики, анимации, моделирования, рендеринга и других компьютерных визуальных эффектов. Она предоставляет широкий спектр инструментов и функций для работы с трехмерными моделями, текстурами, освещением, анимацией и симуляциями. «Blender» поддерживает различные техники моделирования, включая полигональное моделирование, скульптинг, ретопологию, текстурирование и многое другое. Данный инструмент набирает популярность среди современных разработчиков игр, кинокомпаний и VFX-студий. Интерфейс данного приложения изображен на рисунке 21.

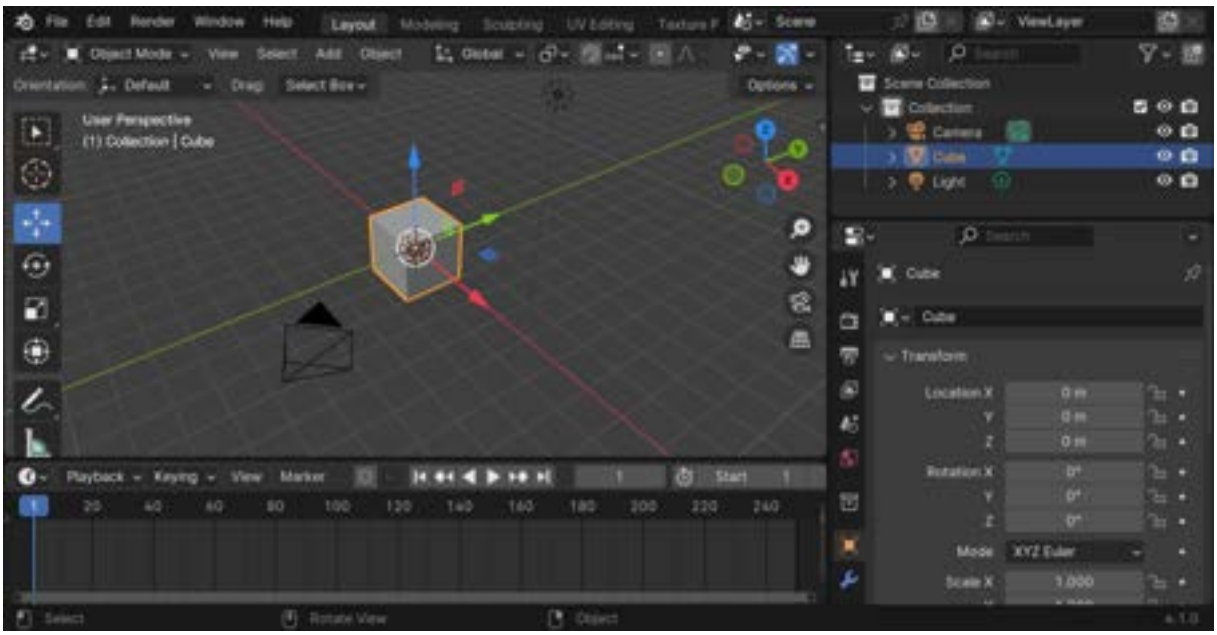

Рисунок 21 – Интерфейс программы «Blender»

2. Mixamo – это онлайн-сервис компании «Adobe», который предоставляет инструменты для создания и настройки трехмерной анимации и персонажей (рисунок 22). С помощью «Mixamo» пользователи могут выбирать готовые трехмерные модели персонажей, анимации движений, а также применять различные настройки, такие как изменение размера и пропорций персонажей, добавление анимации и многое другое.

Рисунок 22 – Главная страница «Mixamo.com»

«Mixamo» также предлагает возможность загружать собственные модели и анимации для дополнительной настройки. Данный онлайн-сервис особенно полезен для разработчиков игр, аниматоров и дизайнеров, которые работают с трехмерными объектами.

Для разнообразия окружающих объектов и дополнительной реалистичности создадим в программе «Blender» несколько моделей. Данные модели будут служить «наполнением» игрового уровня.

Создадим несколько домов разной величины, как изображено на рисунке 23.

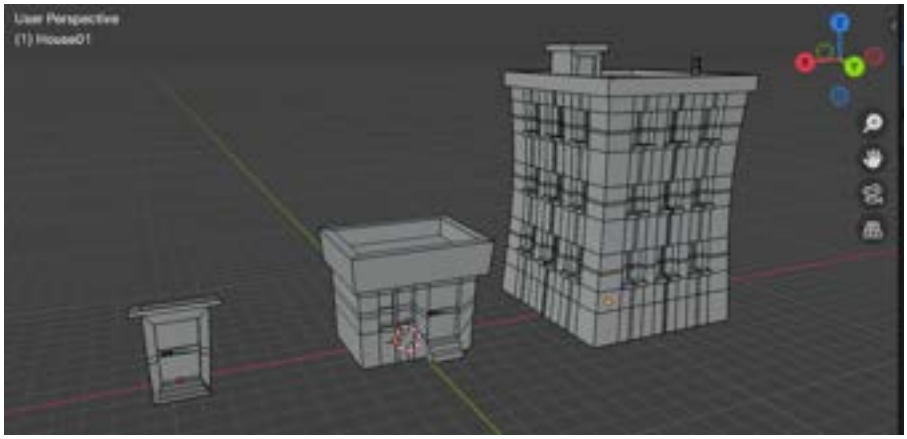

Рисунок 23 – Здания среднего и малого размеров

Помимо малых домов, сделаем здания большого размера (рисунок 24).

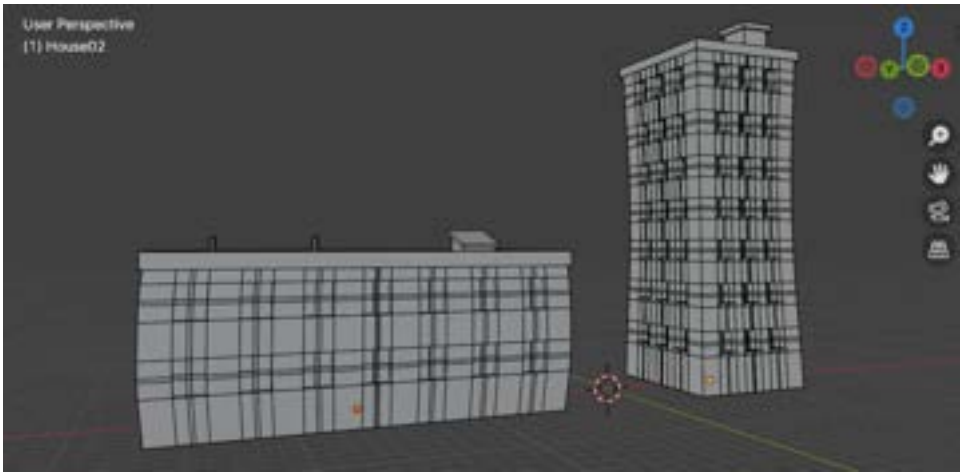

Рисунок 24 – Дома большого размера

Построим в редакторе иные объекты окружения для придания атмосферы игры: столб, дорожный знак, светофор и ограждения (рисунок 25).

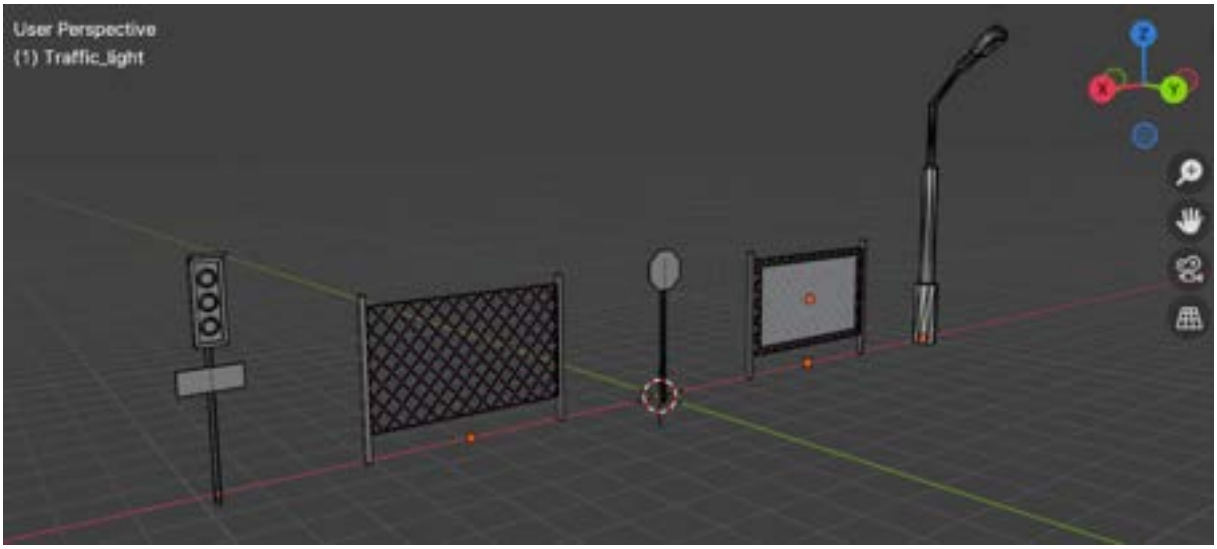

Рисунок 25 – Другие объекты окружения

Для меню игры спроектируем объект с динамическим освещением – рекламный столб с надписью нашего проекта, который будет символизировать наш придуманный игровой мир (рисунок 26).

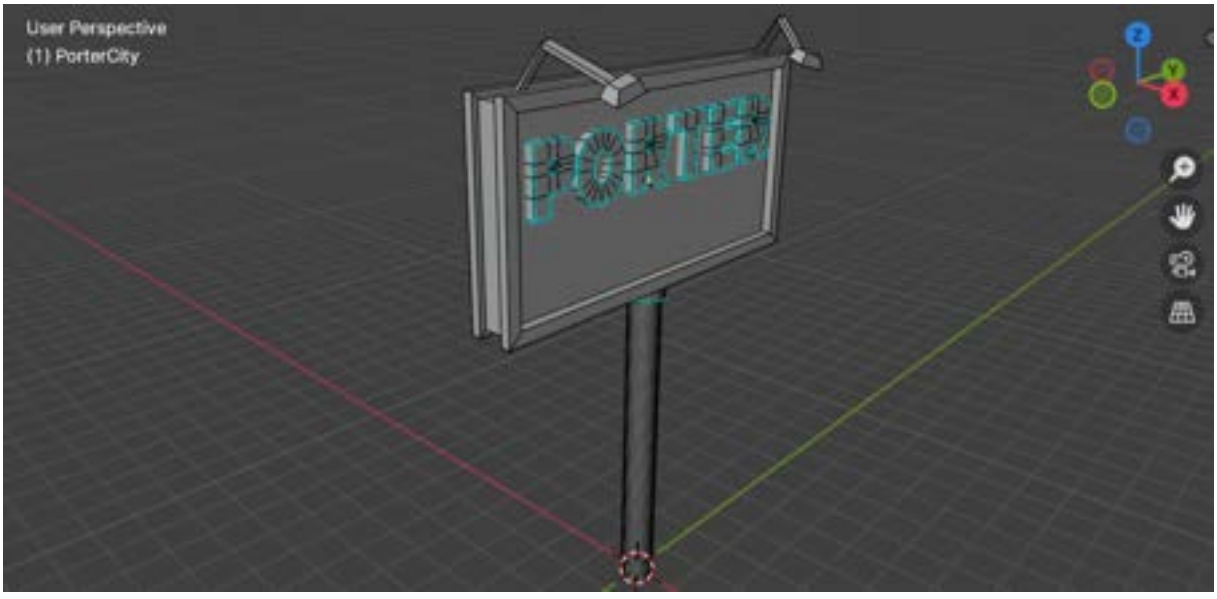

Рисунок 26 – Декоративный объект с надписью проекта

После проектирования моделей в программе «Blender» мы можем разукрасить их. Для простоты визуализации зададим моделям монотонный серый цвет и затемним некоторые места для объема (рисунок 27).

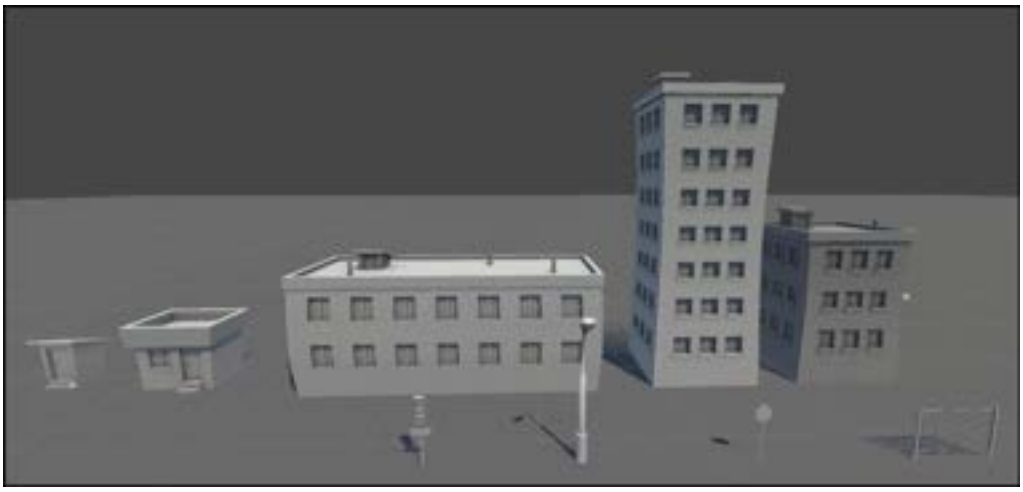

Рисунок 27 – Текстурирование декоративных объектов

Для некоторых объектов можно добавить на текстуру тени и искусственные источники света (рисунок 28). Данные методы текстурирования позволят сделать объекты игры минималистичными и приятными для восприятия.

Рисунок 28 – Объект с искусственным затенением и светом.

2.3. Проектирование элементов интерфейса и меню игры

Интерфейс (UI) – неотъемлемая часть любого программного продукта, не только игрового. От него зависит восприятие игры, интерес пользователей к ней и удобство игрового процесса.

В проектирование интерфейса входят следующие шаги:

– создание дизайна для меню: необходимо продумать внешний вид меню и расстановку кнопок, вид элементов интерфейса и шрифт;

– создание элементов для загрузочного листа: полоска загрузки уровня или подвижный элемент для обозначения процесса, фоновое изображение, совет игроку или один из полезных фактов;

– создание элементов вывода информации во время игрового процесса: шрифт, кнопки, списки, иконки и декоративные элементы.

Важно понимать, что для удачного результата необходимо не отклоняться от общего дизайна игры, учитывать стиль и смысловую идею проекта: объекты должны быть одного стиля и одной цветовой гаммы. Фоновое изображение должно соответствовать тематике и отображать какие-то особые элементы игры, либо фоновым отображением может служить трехмерная сцена, где будет располагаться ключевой объект игры с высокой детализацией.

Для проектирования дизайна некоторых растровых объектов будем использовать бесплатный растровый редактор «Krita». Данный графический редактор является профессиональным бесплатным инструментом для создания различных изображений, иконок и комиксов.

1. «Играть» – кнопка, которая будет переносить нашего игрока на сцену с игровым процессом.
2. «Об игре» – кнопка, которая будет открывать информацию об игре и разработчике.
3. «Выход» – кнопка для выхода из игры.

Помимо этого необходимо добавить название, логотип и объекты наполнения для отображения главной сути нашей игры (рисунок 29).

Рисунок 29 – Проектирование главного меню игры

После проектирования основного меню нам необходимо создать лист информации «Об игре», на котором будет отражено управление и разработчик, результат изображен на рисунке 30.

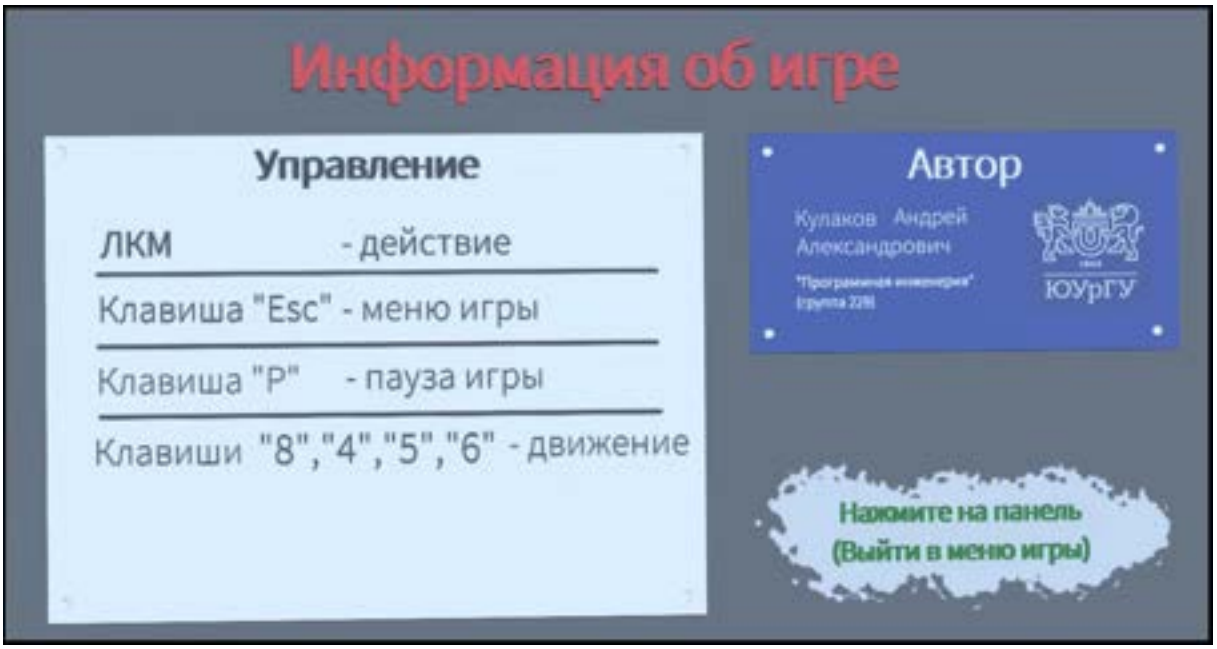

Рисунок 30 – Проектирование окна «Об игре»

Данная информация будет изображена на трехмерном объекте, который будет располагаться в нашем объемном меню (рисунок 31).

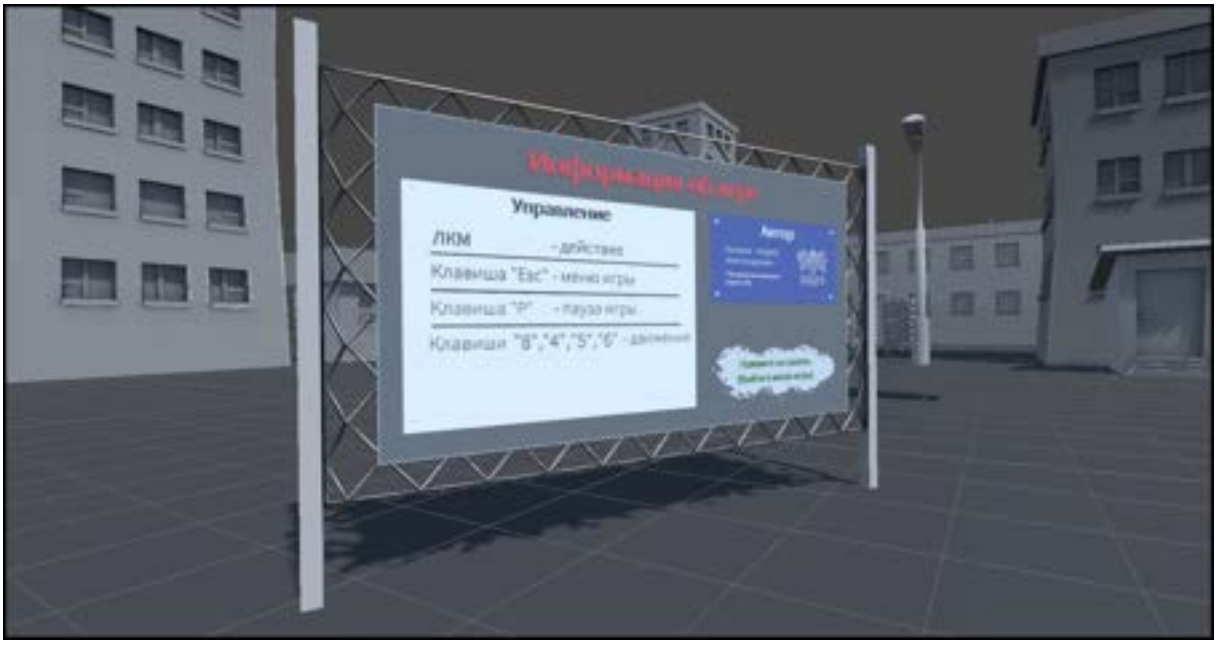

Рисунок 31 – Игровой стенд с информацией «Об игре»

Загрузочный лист – это этап перехода из меню в саму игру. Он нужен для того, чтобы занять время подгрузки объектов игры: 3D-моделей, текстур, изображений, информации, скриптов и необходимых файлов. Данный процесс может быть длительным, поэтому загрузочные листы помогают игрокам понять, что игра активно загружается и готовится к началу, а также предоставляют возможность отобразить логотипы разработчиков, подсказки, советы, рекламу или просто изображения из игры.

Так как на сцене не очень много объектов, и загрузочное время будет составлять всего 5–10 секунд, то будем использовать плавное затухание, выведем небольшой текст цели игры и надпись «Загрузка…».

2.4. Проектирование главных игровых объектов игры

Главные игровые объекты – это объекты, которые должны взаимодействовать с самим игроком или игровым персонажем. Данные объекты могут варьироваться в зависимости от игры, так как каждая игра имеет

свои собственные игровые механики. Например, в разных играх это могут быть персонажи, оружие, предметы инвентаря, препятствия и цели.

Разновидности главных игровых объектов могут быть следующими:

– персонажи: управляемые персонажи, NPC, боссы;

– оружие и предметы: различные виды оружия, амуниция, зелья, ключи, монеты;

– препятствия: ловушки, стены, вода, огонь;

– цели и задачи: объекты, которые нужно собрать, места, куда нужно добраться, задачи, которые нужно выполнить.

Интерактив игровых объектов зависит от их типа. Некоторые игровые объекты могут взаимодействовать с помощью простых действий, в то время как другие могут требовать решения головоломок, использования специальных способностей или даже ведения диалогов с персонажами.

Для проекта и задуманной идеи необходимо создать модель персонажа, который будет изображен как грузчик. С помощью программы «Blender» создадим антропоморфного персонажа, наложим на него текстуру и добавим объемные тени, как показано на рисунке 32.

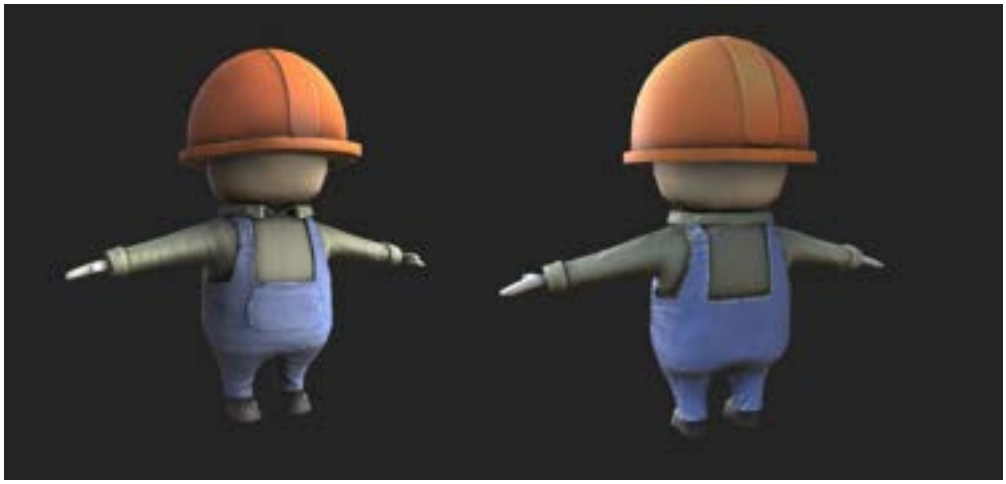

Рисунок 32 – Моделирование игрового персонажа

После создания визуальной оболочки персонажа, необходимо создать ему скелет с помощью программы «Blender». Скелет позволит «оживить» персонажа, и добавить ему в игре анимации ходьбы. Для удобной

работы с человеческим скелетом в программе для трехмерного моделирования присутствует аддон «Rigging: Rigify», но для этого его необходимо включить в настройках программы (рисунок 33).

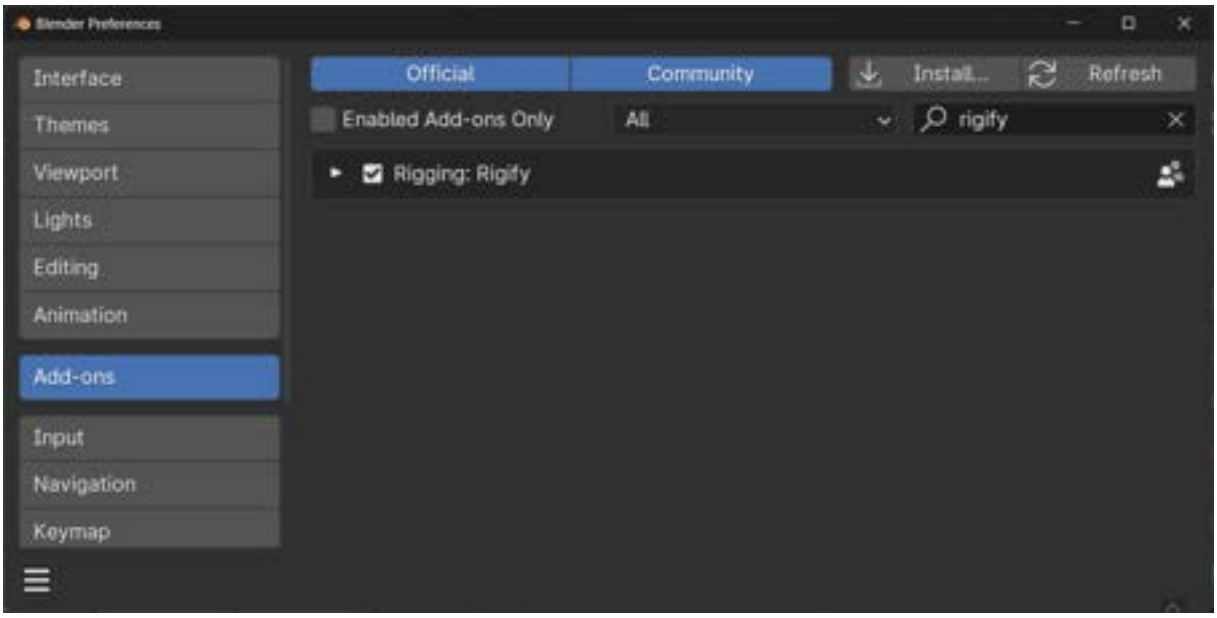

Рисунок 33 – Моделирование игрового персонажа

Теперь в меню добавления объектов выберем «Human (Meta-Rig)» и подстроим скелет под модель грузчика как показано на рисунке 34.

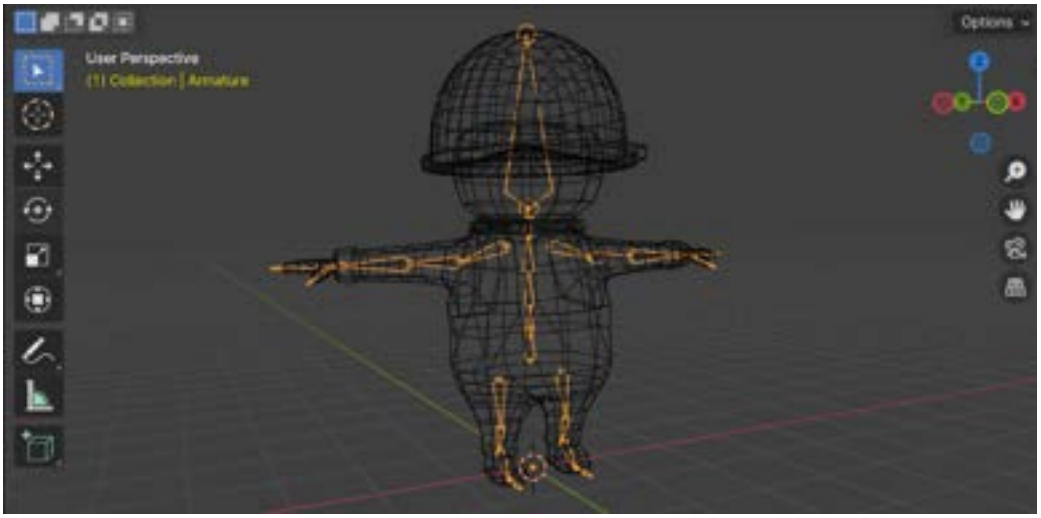

Рисунок 34 – Добавление скелета персонажу

Теперь персонажа необходимо импортировать в онлайн-сервис «Mixamo», для того чтобы подобрать для него необходимые анимации ходьбы и бега. Для работы в «Mixamo» необходимо создать учетную за-

пись и импортировать модель персонажа в формате «fbx». Работа с данным сервисом показана на рисунке 35.

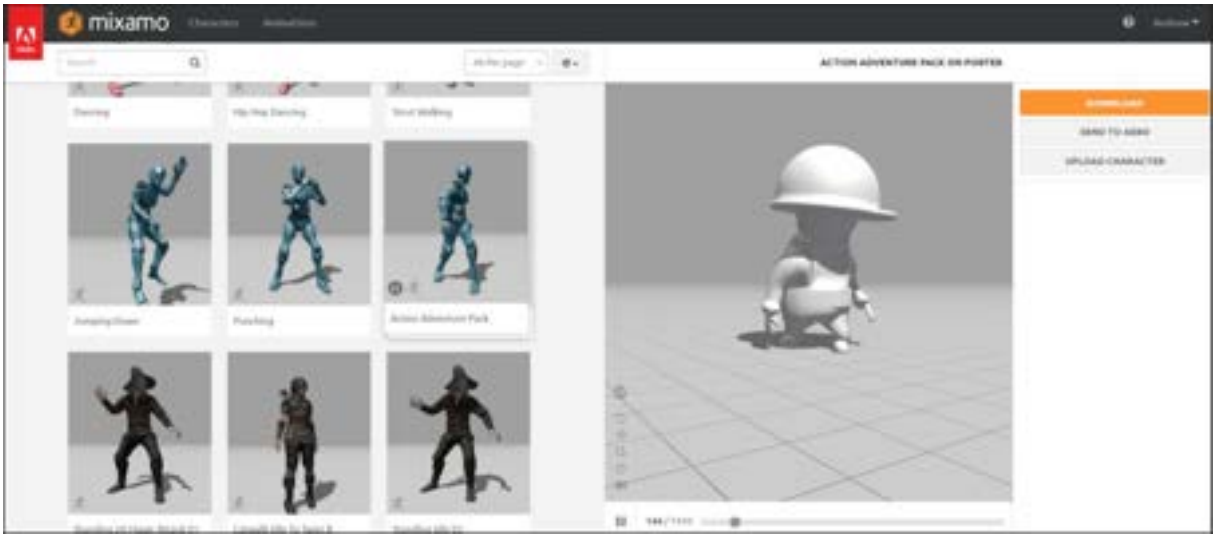

Рисунок 35 – Работа с сервисом «Mixamo»

После того, как мы создали персонажа, спроектируем модель места, в котором будут происходить действия игры. Для удобства навигации игрока и наблюдения за происходящим, не будем моделировать крышу помещения, но создадим места для будущих конвейеров. Модель спроектированного склада представлена на рисунке 36.

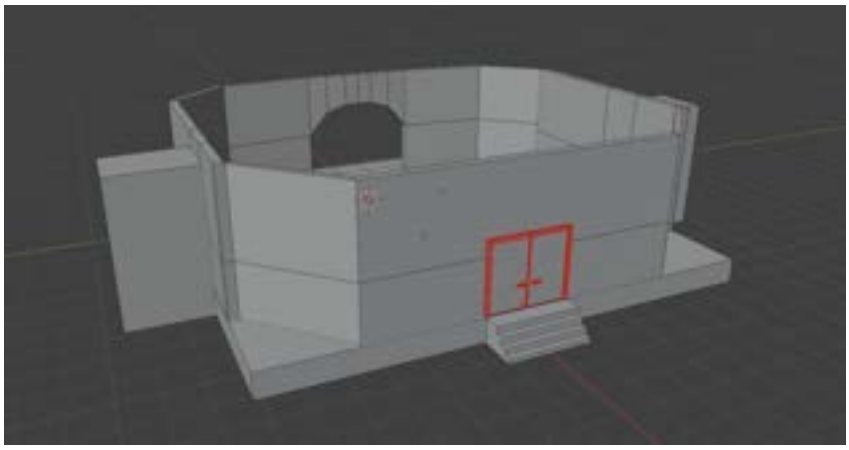

Рисунок 36 – Модель склада

После проектирования трехмерной модели, создадим текстуру кладки кирпичей, цементных плит, наложим на стены декоративные надписи, плакаты и таблички, как показано на рисунке 37.

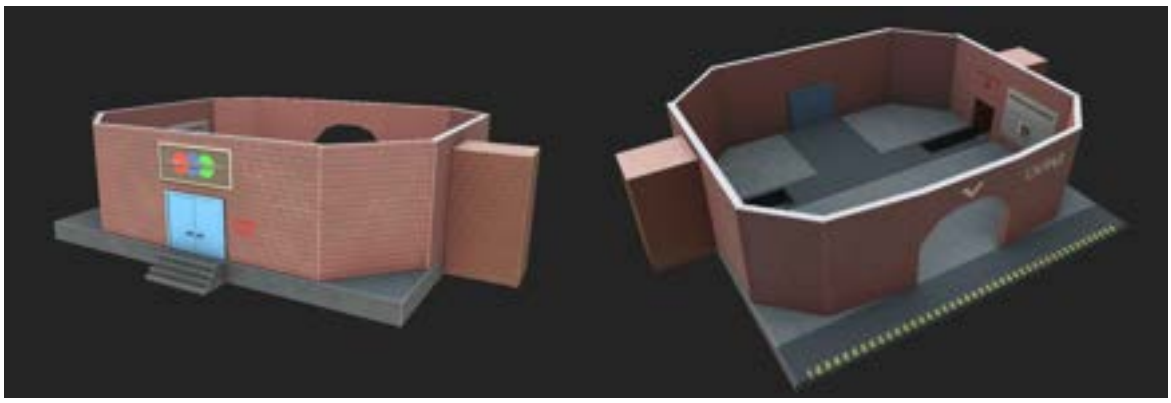

Рисунок 37 – Модель склада с текстурой

На данном объекте будут располагаться программные «коллайдеры», которые не будут давать возможность проходить через стены персонажам, изменяемые надписи со счетом очков и источники света, для равномерного освещения.

Для продуманных в помещении ям, создадим конвейеры, которые буду перемещать груз из точек появления в наше помещение. Чтобы данный объект был с корректной анимацией на текстуре ленты, нужно соблюдать бесшовность текстуры и последовательность сегментов развертки модели (рисунок 38).

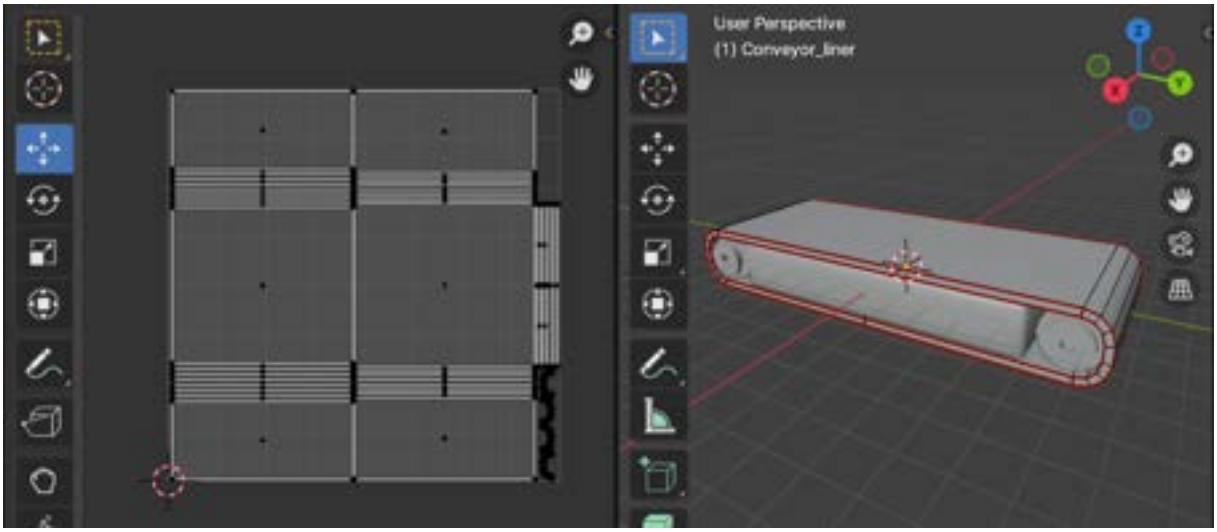

Рисунок 38 – Модель конвейера и изображение развертки

Данная модель состоит из двух текстур, одна из частей динамическая, другая статичная. Модель конвейера с текстурой изображена на рисунке 39.

Рисунок 39 – Модель конвейера с текстурой

Теперь спроектируем цель в виде куба и наложим текстуру коробки (рисунок 40). Данный объект будет взаимодействовать с окружающей игровой средой и персонажами с помощью компонента «Rigidbody» в движке «Unity».

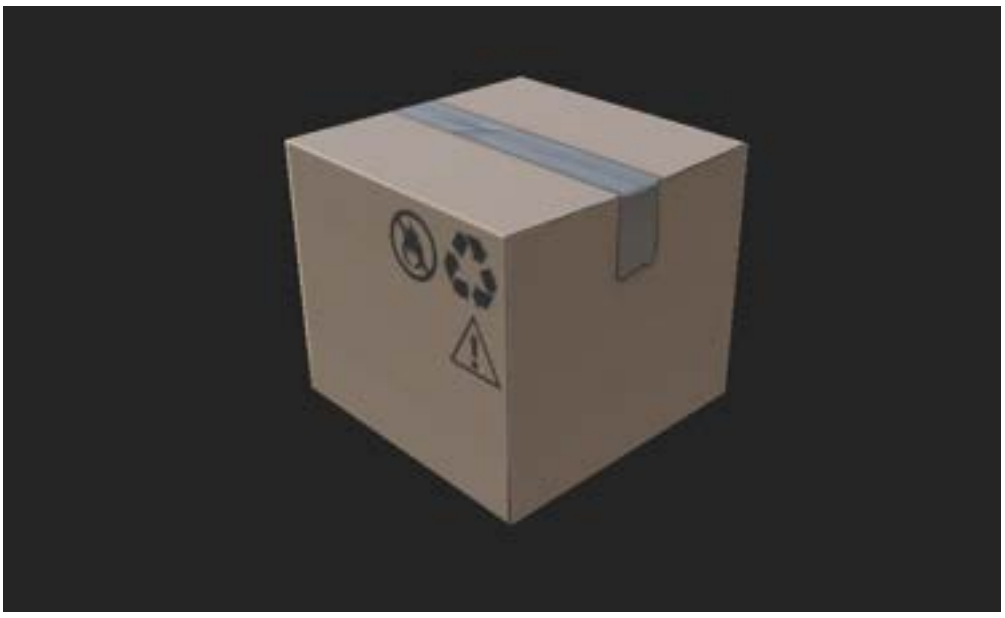

Рисунок 40 – Модель коробки

После реализации цели, необходимо создать модель грузовика, в который необходимо складировать данные коробки. Объект будет иметь внутреннее помещение, в которое можно будет пройти персонажам.

Реализованная трехмерная модель машины изображена на рисунке 41.

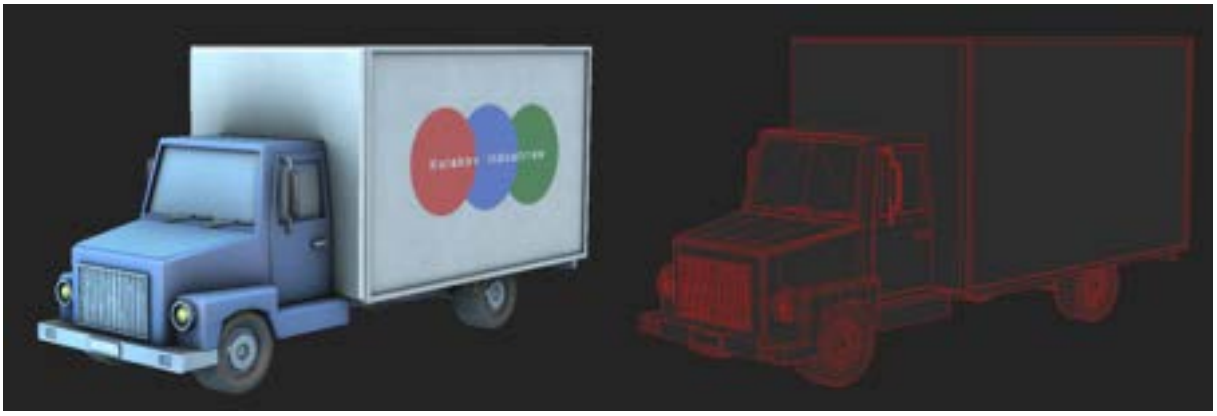

Рисунок 41 – Модель грузовика

После создания всех необходимых моделей, необходимо перенести наши ресурсы в игровой движок «Unity», задать материалы и указать текстуры. Помимо этого, каждому спроектированному объекту нужно задать компонент «Collider» для их физического взаимодействия друг с другом. После того, как все игровые объекты были реализованы, разместим их на сцене как показано на рисунке 42.

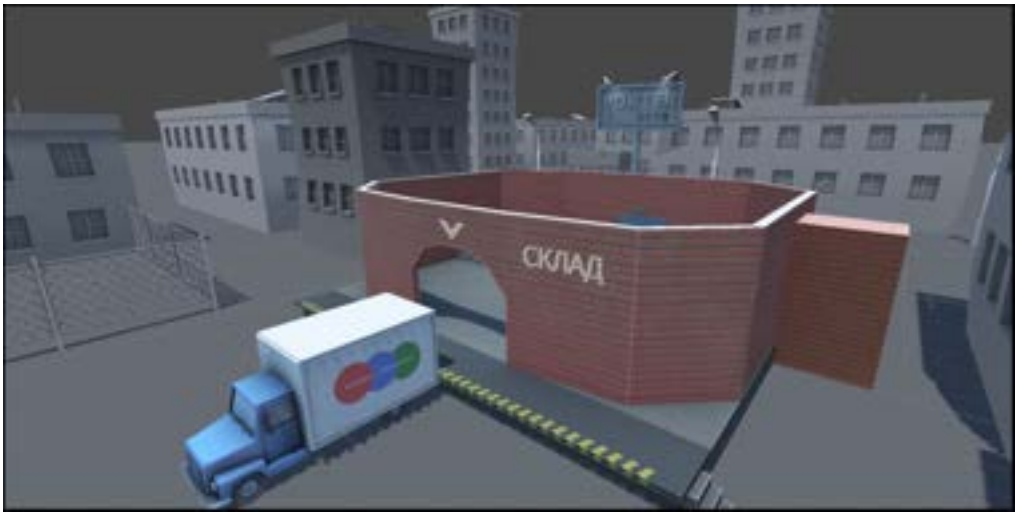

Рисунок 42 – Спроектированная игровая сцена

Внутри спроектированного помещения разместим персонажей, конвейеры и добавим динамические надписи, которые будут отражать счет, время и таблицу рекордов как показано на рисунке 43.

Рисунок 43 – Объекты внутри игрового помещения

После импорта объектов и размещения их на сцене, необходимо создавать объекты «Префабы».

Префаб в Unity – это объект-шаблон, содержащий настроенные компоненты, свойства и другие данные объекта, который можно многократно использовать в проекте. Применение префабов делает процесс создания и управления объектами в Unity более эффективным. Префабы позволяют быстро создавать и изменять множество однотипных объектов в игре, обеспечивая единообразие настроек и упрощая разработку. Данные объекты понадобятся для обучения будущего интеллектуального агента.

2.5. Проектирование игровой механики

На сцене располагается три подвижных объекта и один статичный: персонаж игрока, персонаж интеллектуального агента, груз, который будут двигать и машина, в которую будут складировать груз. Задача агента, будет заключаться в том, чтобы научится двигать куб к цели и делать это с

минимальными ошибками, и как можно скорее выполнять свою задачу, в это же время игроку необходимо помогать «игровому товарищу» в перемещении груза к машине. За выполнение задачи интеллектуальный агент будет получать вознаграждение, а счет игрока будет увеличиваться.

Для начала создадим префаб персонажа игрока, подключим к нему компонент «Animator» со структурой анимаций, показанной на рисунке 44.

Рисунок 44 – Компонент «Animator»

В данной структуре компонента «Animator» можно увидеть три состояния персонажа: статичное (персонаж стоит на месте), состояние ходьбы (персонаж перемещается с небольшой скоростью) и состояние бега (персонаж достиг максимальной скорости). У каждой анимации есть двунаправленная связь, которая позволит менять анимации в любом направлении.

Составим класс на языке C# под названием «PlayerPorter», который описан в приложении А. Параметры «Main Camera», «Max Speed», «Axes», «Sensitivity» отвечают за управление камерой, скорость перемещения и чувствительность управления камерой курсором мыши по оси X и Y.

Вложим в префаб персонажа игрока камеру, подключим C#-класс, добавим компоненты «Character Controller», «Rigidbody» как показано на рисунке 45.

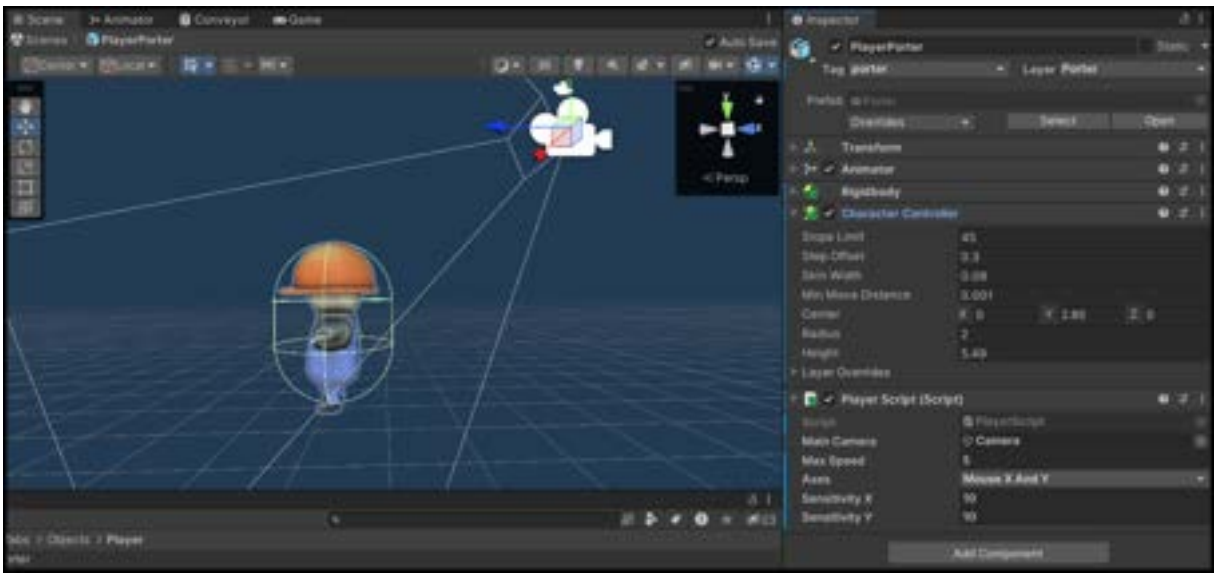

Рисунок 45 – Префаб «PlayerPorter»

В префабе «Porter» создадим объект интеллектуального агента, для этого необходимо добавить персонажу объект «Empty» и задать ему компонент «Ray Perception Sensor», настроить данный компонент, указав теги объектов, которые необходимо видеть агенту. После данных настроек добавим префабу компоненты «Collider», «Animator» и «Rigidbody» для физического взаимодействия с окружающей средой.

Реализуем класс C# под названием `PorterAgent`, который указан в приложении Б. Задача данного класса заключается в передаче расположения объектов сцены в модель интеллектуального агента. Помимо этого, в классе данного объекта описан алгоритм перемещения персонажа по нажатию стрелок клавиатуры, которые будут программно нажиматься агентом.

При добавлении данного класса в «инспектор» персонажа, появится панель настройки машинного обучения и список с указанием ссылок на объекты взаимодействия и объекты появления новых целей. Настроим данные параметры как показано на рисунке 46.

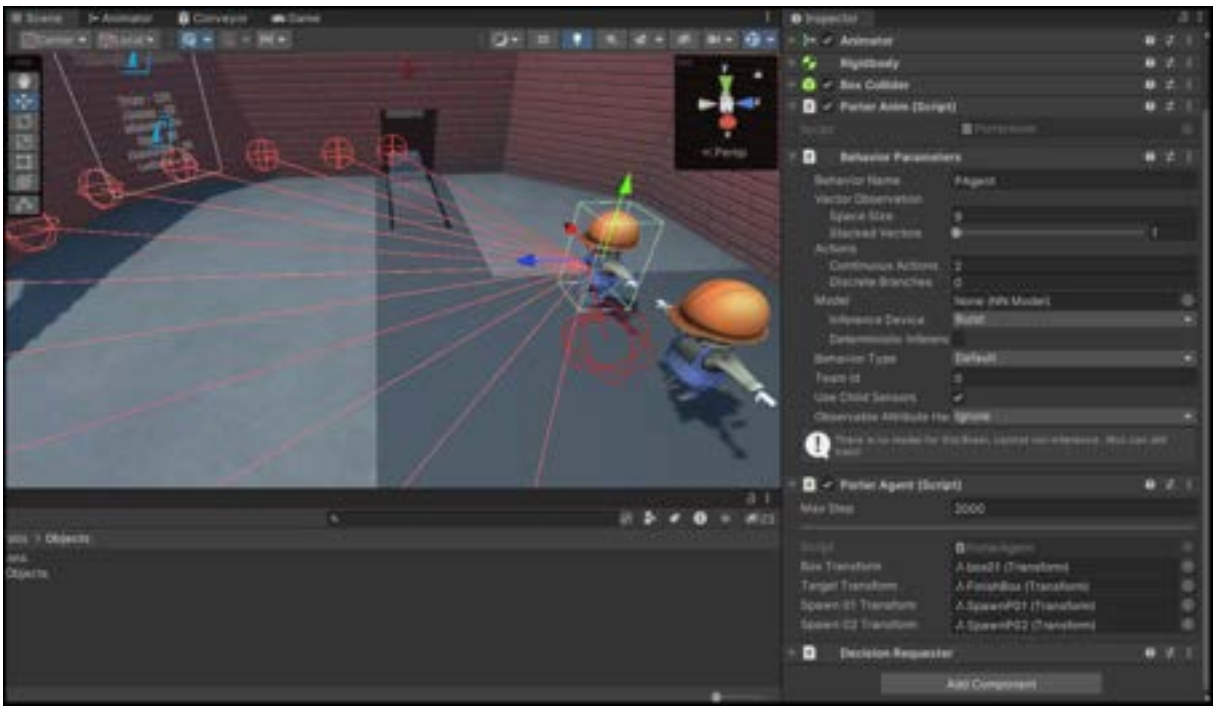

Рисунок 46 – Префаб «Porter»

Объекту груза (префаб «box») необходимо задать класс, который получает ссылку на объект интеллектуального агента и с помощью функции `AddReward()` поощряет персонажа за взаимодействие с коробкой, но в то же время штрафует его за выполнение простых ходов бездействия или за столкновение коробки со стенами. За счет такого подхода к штрафам, будем заставлять интеллектуального агента быстрее находить решение и искать пути вознаграждения, помимо этого, данное условие будет заставлять персонажа быть аккуратным к грузу, не сталкивать его со стенами, чтобы коробка не застревала. Вознаграждение искусственного игрока будет происходить за касание им груза.

В листинге 1 приведен код класса `TargetBox`, который описан в объекте «box».

Листинг 1 – Код класса `TargetBox`

```
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using Unity.MLAgents;
using UnityEngine;
public class TargetBox : MonoBehaviour
{
```

```
    public PorterAgent pagent;
    public float conveyor_delay = 0f;
    private void Update()
    {
        pagent.AddReward(-0.00001f);
    }
    void OnCollisionEnter(Collision col)
    {
        if (col.gameObject.CompareTag("porter"))
        {
            pagent.AddReward(+0.01f);
        }
        if (col.gameObject.CompareTag("wall"))
        {
            pagent.AddReward(-0.01f);
        }
    }
}
```

Теперь необходимо создать классы `Target` и `LVLTimer` для объекта получения перемещаемого груза. Зададим систему вознаграждения таким образом, чтобы игровой интеллектуальный агент получал значительно больше вознаграждения при доставлении коробки в пункт принятия грузов, чем при взаимодействии с грузом. Данная последовательная логика поощрения необходима для мотивации персонажа, чтобы он стремился перемещать груз в заданную точку. Класс `Target`, который описан в листинге 2, предназначен для вознаграждения игрового агента, если тот перенесет груз в машину.

Листинг 2 – Код класса `Target`

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SocialPlatforms.Impl;
public class Target : MonoBehaviour
{
    public TMP_Text _score;
    public TMP_Text _timer;
    public Transform spawn01Transform;
    public Transform spawn02Transform;
    public PorterAgent pagent;
    public GameObject box;
    private int Score = 0;
    private void Update()
    {
        _score.text = "Счёт: " + Score;
    }
    public void OnTriggerStay(Collider other)
    {
        if (other.TryGetComponent<TargetBox>(out TargetBox box))
```

```
        {
            if (_timer.text != "Время кончилось!")
            {
                Score += 1;
            }
            pagent.EndEpisode();
            pagent.AddReward(+5f);
        }
    }
}
```

Для подсчета времени необходим отдельный класс, который будет подсчитывать оставшееся время и выводить результат в текстовое поле на сцене. В данном классе необходимо задать количество секунд, которое для удобства восприятия переводит значение в два числа: количество минут и секунд. При окончании таймера будет выводиться текст «Время закончилось!» и подсчет очков будет остановлен в классе `Target`. Помимо этого, этот класс будет следить за нажатием некоторых клавиш для того, чтобы поставить игру на паузу или выйти в меню. Данный класс указан в приложении В.

Создадим скрипт под названием `Conveyors`, в котором будет метод для перемещения стоящих на конвейере предметов (листинг 3).

Листинг 3 – Код класса `Conveyors`

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Conveyors : MonoBehaviour
{
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag == "porter"){
            other.gameObject.GetComponent<Rigidbody>().
                  MovePosition(other.transform.position
                  + this.transform.forward * -1.5f * Time.deltaTime);
        }
        if (other.gameObject.tag == "box"){
            other.gameObject.GetComponent<TargetBox>().
                        conveyor_delay -= Time.deltaTime;
        if (other.gameObject.GetComponent<TargetBox>().conveyor_delay < 0)
            {
                other.gameObject.GetComponent<Rigidbody>().
                    MovePosition(other.transform.position +
                    this.transform.forward * -1.5f * Time.deltaTime);
            }
        }
    }
}
```

У ленты конвейера необходимо задать динамический материал, который будет циклично смещать расположенную на нем текстуру. Данный материал необходимо создать с помощью инструмента Unity Shader Graph, как показано на рисунке 47.

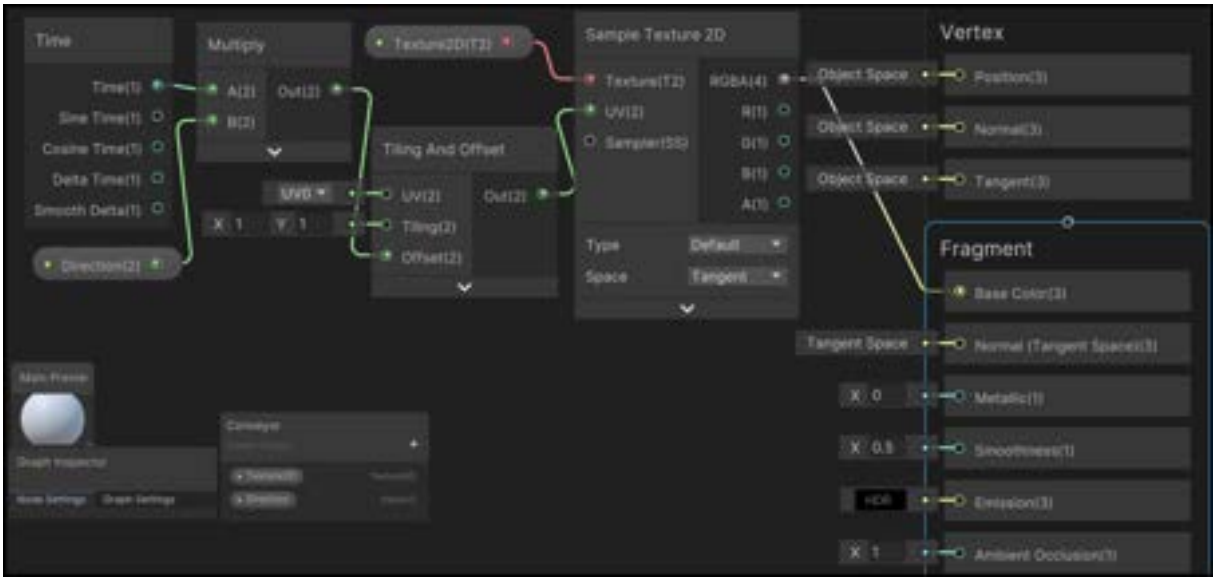

Рисунок 47 – Динамический материал конвейера

2.6. Обучение интеллектуального агента

После создания игровых шаблонов, необходимо создать отдельную закрытую сцену игры без декораций. Это позволит производить обучение интеллектуального агента быстрым, максимально производительным и наглядно простым.

Для данной задачи создадим сцену под названием «TeachScene», а из главных действующих объектов игры составим «общий префаб». В нем будут находиться объекты помещения, персонаж, конвейеры, груз и грузовик. Это необходимо для будущей изоляции игровых комнат и удобного копирования на сцене. Префаб, который объединяет в себе все объекты изображен на рисунке 48. Заметим, что на сцене отсутствует персонаж игрока, потому что для данной задачи он не понадобится.

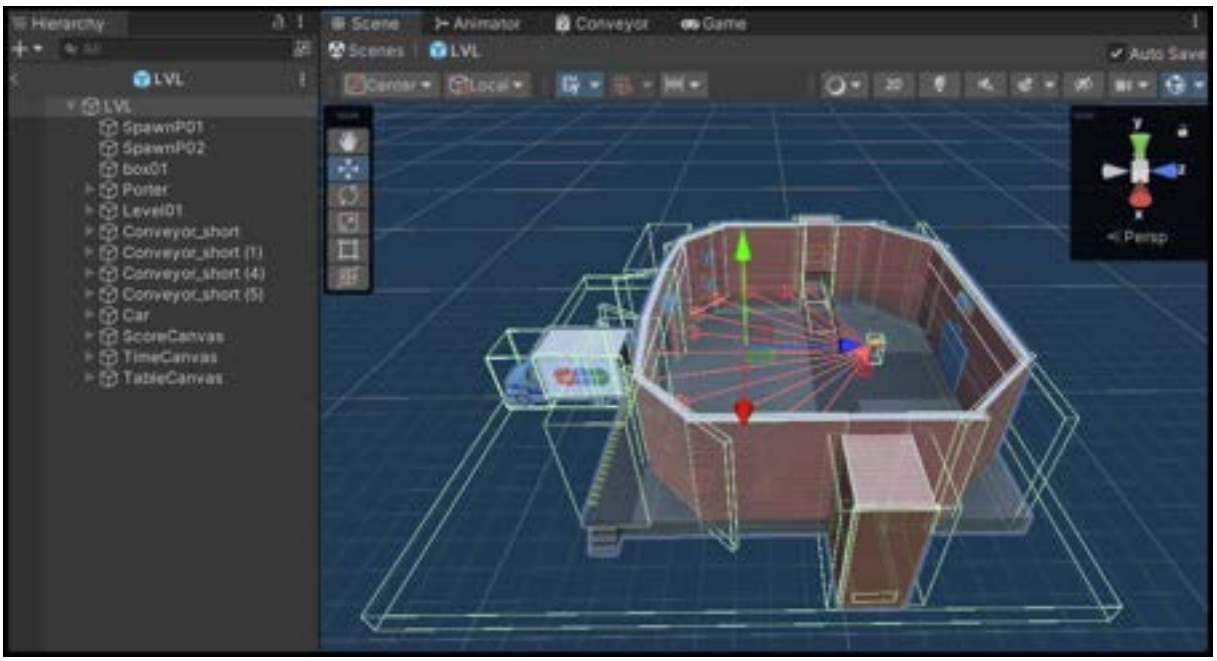

Рисунок 48 – Префаб для обучения интеллектуального агента

После создания данного префаба, расположим 15 копий данного объекта на сцене «TeachScene». У каждого объекта на сцене будет использоваться своя локальная система координат, благодаря этому каждый интеллектуальный агент будет воспринимать себя единственным на сцене и искать только свой объект груза (рисунок 49).

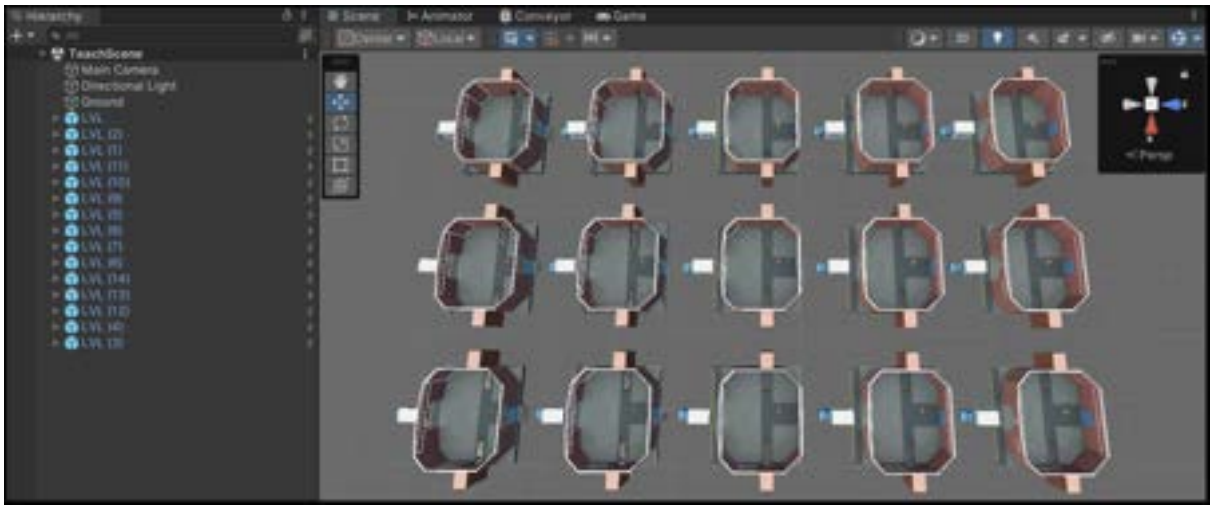

Рисунок 49 – Префаб для обучения интеллектуального агента

Данное клонирование необходимо для ускоренного одновременного обучения одного мозга интеллектуального агента.

После подготовки сцены, необходимо создать папку «Config» в проекте Unity и создать файл «PushBlock.yaml». Данный файл будет содержать в себе конфигурацию обучения, которую рекомендует компания Unity (листинг 4).

Листинг 4 – Параметры в файле «PushBlock.yaml»

```
behaviors:
  PAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 1500000
    time_horizon: 64
    summary_freq: 50000
```

Данная конфигурация позволит расширить максимальное количество шагов для обучения (параметр `max_steps`) до двух миллионов и более гибко настроить остальные параметры, что существенно отличается от стандартных параметров обучения, которые предустановленны в Unity ML-Agents.

Теперь запустим «Командную строку» и введем туда следующую консольную команду, как показано на рисунке 50.

Рисунок 50 – Консольная команда для запуска обучения

Данная команда запустит процесс ожидания и попросит разработчика запустить проект. После запуска приложения интеллектуальный агент начнет самостоятельно двигаться и анализировать окружающую среду. Скорость процесса обучения интеллектуального агента зависит от того, с помощью какого устройства будут обрабатываться вычисления. Выбрать устройство можно путем изменения параметра «Inference Device» у префаба «Porter».

После окончания процесса, консоль выведет сообщение об успешном завершении. Узнать подробности о прогрессе обучения можно с помощью сервиса «TensorBoard», для этого необходимо в консоль ввести команду, которая изображена на рисунке 51. Результатом команды будет сгенерированная ссылка.

Рисунок 51 – Консольная команда для запуска TensorBoard

В данном сервисе можно увидеть все результаты обучения в виде графиков (рисунок 52).

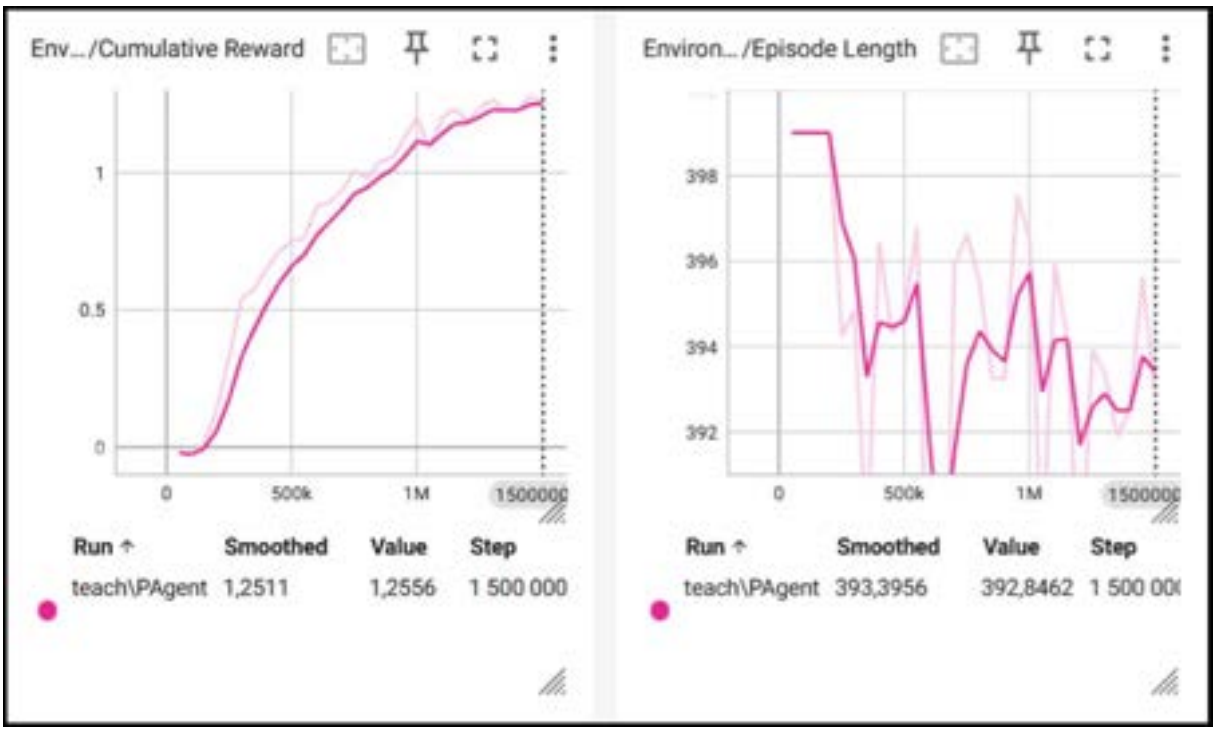

Рисунок 52 – Результаты TensorBoard

Помимо этого можно подробно узнать, сколько интеллектуальный агент получал вознаграждения с течением времени, прогресс ошибок и какие методы поощрения он больше предпочитал получать при обучении.

В папке «results» (которая располагается в корневой папке проекта) появятся результаты обучения модели. В результатах находится файл формата «onnx». Данный файл необходимо перенести в папку «Assets» нашего проекта и в параметре «Model» префаба «Porter» указать его. После этого можно дообучить готовый мозг интеллектуального агента, запустив повторно процесс обучения, но уже под другим названием запуска. После окончания процесса, необходимо заново зайти в папку «results» и перенести новый файл обученной модели.

После удовлетворительного процесса обучения необходимо подключить модель к интеллектуальному агенту, параметр «Behavior Type» переключить на «Inference Only» и убрать в параметрах класса количество максимальных шагов. Это позволит использовать обученную модель в проекте.

Последним этапом сделаем клонов нашего интеллектуального субъекта и расположим их как показано на рисунке 53.

Рисунок 53 – Клонированные интеллектуальные агенты

Каждая копия обученного агента будет действовать самостоятельно и двигать коробку в машину.

После данных действий, когда проект стал готов для использования любым пользователем, необходимо задать иконку для ярлыка приложения, сохранить настройки света для каждой сцены и собрать проект.

Вывод по второму разделу

В данной главе успешно реализовали приложение, начав с установки всех необходимых компонентов для работы с Unity ML-Agents, включая Python, Tensorflow, Pytorch, движок Unity и пакет для работы с машинным обучением Unity ML-Agents. Затем перешли к проектировке объектов игры, создав декоративные дома, столбы, указатели и персонажей грузчиков для минималистичной имитации города.

Далее была спроектирована игровая логика, разработан код для игровых объектов и управления персонажем. На заключительном этапе главы осуществлено обучение игрового интеллектуального агента, который успешно выполнял задачу толкания груза в указанную точку.

3. ТЕСТИРОВАНИЕ

После разработки игрового приложения было произведено функциональное тестирование в целях проверки реализованности и проверки корректной работы функций приложения. Набор функциональных тестов представлен в таблице 1.

Таблица 1 – Тестирование разработанного приложения

№	Название теста	Шаги	Ожидаемый результат	Успешное завершение теста
1	Работа функции «Выход» из приложения.	1. Открыть программу. 2. Нажать на кнопку «Выход».	Приложение должно закрыться.	Да.
2	Работа функции «Об игре».	1. Открыть программу. 2. Нажать на кнопку «Об игре». 3. Нажать на панель информации и выйти в меню.	Приложение должно открывать информацию об игре.	Да.
3	Работа функции кнопки «Играть»	1. Открыть программу. 2. Нажать на кнопку «Играть».	Приложение должно осуществлять переход на сцену с игровым процессом.	Да.
4	Осуществление выхода из процесса игры.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Нажать на клавишу клавиатуры «Esc». 4. Нажать на кнопку «Выход».	Приложение должно закрыться.	Да.
5	Осуществление паузы во время процесса игры.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Нажать на клавишу клавиатуры «P».	Игровой процесс должен остановиться.	Да.
6	Управление игровым персонажем во время игры.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Нажать на клавишу клавиатуры «P».	Игрок должен иметь возможность управлять своим персонажем.	Да.

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат	Успешное завершение теста
7	Взаимодействие игрока с игровым миром.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Толкнуть появившийся груз. 4. Толкнуть другого персонажа. 5. Попробовать пройти за пределы игрового уровня.	Игрок должен оставаться в пределах уровня и взаимодействие с игровыми объектами должно быть корректным.	Да.
8	Игровое время идет корректно.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Посмотреть на игровое табло «Время». 4. Дождаться надписи «Время кончилось!».	Время игры должно корректно подсчитываться и кончаться на игровых часах.	Да.
9	Начисление игровых очков.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Управляя персонажем протащить коробку в поле для начисления очков. 4. Посмотреть на игровое табло «Счет». 5. Дождаться окончания времени. 6. Перетащить ещё одну коробку.	Очки игрока должны начисляться корректно и по правилам игры.	Да.
10	Взаимодействие интеллектуального объекта с игровым миром.	1. Открыть программу. 2. Нажать на кнопку «Играть». 3. Наблюдать за игровым персонажем.	Игровой персонаж должен двигаться, иметь анимации и стараться перетащить груз в игровое поле для начисления очков.	Да.

Вывод по третьему разделу

В данной главе было произведено функциональное тестирование всех элементов игры и игровых механик.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана компьютерная игра с интеллектуальным поведением субъектов на игровом движке Unity 3D. С помощью плагина Unity ML-Agents удалось добиться успеха в разработке данного проекта и сделать игру интересной. При этом были решены следующие задачи:

1) рассмотрены программные продукты и готовые игровые движки;
2) выявлены их возможности, достоинства и недостатки;
3) рассмотрен инструмент Unity ML-Agents;
4) проанализированы публикации, в которых описано применение нейронных сетей и машинного обучения в игровых приложениях;
5) спроектирован игровой проект, который будет использовать нейронную сеть и машинное обучение.

Данный проект может быть всячески улучшен, в игру могут быть добавлены различные режимы игрового процесса, в которых будет различаться цель. Помимо этого может быть добавлено множество разновидностей интеллектуальных персонажей, задачи которых будут отличаться. Разработка дополнительных уровней, квестов и задач поможет увеличить продолжительность игры и создать более насыщенный игровой контент для пользователей. Улучшенная графическая составляющая может быть дополнена новыми и улучшенными моделями, игровыми объектами, мир игры можно всячески разнообразить, благодаря чему, игра станет ещё более увлекательной для пользователей.

ЛИТЕРАТУРА

1. Бонд Д.Г. Unity и C#: Геймдев от идеи до реализации. // Питер, 2022. – № 2. – 928 с.

2. Веретехина С. В., Симонов В. Л., Кармицкий К. С. Программирование, тестирование, проектирование, нейросети, технологии аппаратно-программных средств. // ДиректМедиа, 2022. – № 1. – 144 с.

3. Горбаченко В.И. Машинное обучение: учебное пособие. // Ай Пи Ар Медиа, 2023. – № 2. – 217 с.

4. Козлов Д. А., Мясников В. В. Влияние состава наблюдений окружающей среды в задаче приобретения навыков передвижения в трёхмерном пространстве при использовании алгоритмов обучения с подкреплением. // Самарский университет, 2022. – № 1. – 2 с.

5. Лекун Я. Как учится машина: революция в области нейронных сетей и глубокого обучения. // Альпина PRO, 2021. – № 3. – 335 с.

6. Максименкова О.В. Программирование в Unreal Engine 5 для начинающего игродела: основы визуального языка Blueprint. // Бомбора, 2022. – № 2 – 320 с.

7. Машинное обучение и ИИ в разработке игр в 2023 году. [Электронный ресурс] URL: https://platoaistream.com/ru/plato-data/machine-learning-and-ai-in-game-development-in-2023 (дата обращения: 05.02.2024 г.).

8. Машинное обучение в Unity: учим МО-агентов перепрыгивать через стены. [Электронный ресурс] URL: https://habr.com/ru/companies/pixonic/articles/492548/ (дата обращения: 05.02.2024 г.).

9. Павлова А.И. Искусственные нейронные сети: учебное пособие. // Ай Пи Ар Медиа, 2021. – № 3. – 190 с.

10. Сантану, Паттанаяк Глубокое обучение и TensorFlow для профессионалов. Математический подход к построению систем искусственного интеллекта на Python. // Диалектика-Вильямс, 2020. – № 2. – 480 с.

11. Соков И.А. Влияние изменения параметров среды обучения для нейронных сетей в ML-Agents Unity. // Тольяттинский государственный университет, 2023. – № 2. – 14 с.

12. Торн А. Основы анимации в Unity. // ДМК Пресс, 2019. – № 2. – 176 с.

13. Траск Э. Грокаем глубокое обучение. // Питер, 2019. – № 2. – 352 с.

14. Хайкин С. Нейронные сети. // Вильямс, 2008. – № 2. – 1103 с.

15. Харрисон Ф. Изучаем C# через разработку игр на Unity. // Питер, 2021. – № 5. – 400 с.

16. Чару А. Нейронные сети и глубокое обучение: учебный курс. // Диалектика-Вильямс, 2022. – № 2 – 752 с.

17. Шарден Б. Крупномасштабное машинное обучение вместе с Python. // ДМК Пресс, 2018. – № 4 – 358 с.

18. Aizenberg I., Aizenberg N. Multi-Valued and Universal Binary Neurons. // Springer US, 2013. – № 3. – P. 276.

19. Charles D.K. The past, present and future artificial neural networks in digital games. // Ulster university, 2004. – №. 2. – P. 169.

20. Lanham M. Learn Unity ML-Agents – Fundamentals of Unity Machine Learning. // Packt Publishing Ltd, 2018. – №. 1. – P. 204.

21. Neural Networks in Games. [Электронный ресурс] URL: https://studyres.com/doc/3751458/neural-networks-in-games (дата обращения: 05.02.2024 г.).

22. Unity: A General Platform for Intelligent Agents. [Электронный ресурс] URL: https://www.researchgate.net/publication/327570403_Unity_A_General_Platform_for_Intelligent_Agents (дата обращения: 05.02.2024 г.).

23. Unity ML Agents. [Электронный ресурс] URL: https://www.gamedev.net/articles/programming/engines-and-middleware/unity-ml-agents-r5114 (дата обращения: 05.02.2024 г.).

24. Using Machine Learning Agents Toolkit in a real game: A beginner’s guide. [Электронный ресурс] URL: https://blog.unity.com/games/using-machine-learning-agents-in-a-real-game-a-beginners-guide (дата обращения: 05.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Код управления персонажем игрока.

Листинг 1 – Класс `PlayerScript`

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using static UnityEditor.IMGUI.Controls.PrimitiveBoundsHandle;
public class PlayerScript : MonoBehaviour
{
    public GameObject MainCamera;
    Animator porterAnim;
    int last_anim = 1;//1 - idle, 2 - walk 3- run
    public float maxSpeed = 20f;
    public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
    public RotationAxes axes = RotationAxes.MouseXAndY;
    public float sensitivityX = 10F;
    public float sensitivityY = 10F;
    float rotationX = 0F;
    float rotationY = 0F;
    float deltaX = 0f;
    float deltaY = 0f;
    private CharacterController controller;
    private Vector3 playerVelocity;
    Vector3 m_EulerAngleVelocity;
    float pushPower = 8.0f;
    void Start()
    {
        porterAnim = gameObject.GetComponent<Animator>();
        IdlePorter();
        controller = GetComponent<CharacterController>();
    }
    void Update()
    {
        float ForwardBack = 0f;
        float RightLeft = 0f;
        float speed = controller.velocity.magnitude;
        if (speed < 1f)
        {
            if (last_anim != 1)
            {
                last_anim = 1;
                IdlePorter();
            }
        }
        else if ((speed >= 1f) && (speed < maxSpeed / 2))
        {
            if (last_anim != 2)
            {
                last_anim = 2;
                WalkPorter();
            }
        }
        else if (speed >= maxSpeed / 2)
        {
            if (last_anim != 3)
            {
                last_anim = 3;
                RunPorter();
            }
```

```
        }
        if (Input.GetKey(KeyCode.Keypad8))
        {
            ForwardBack = 1f;
            playerVelocity = transform.forward * maxSpeed * ForwardBack;
        }
        else if (Input.GetKey(KeyCode.Keypad5))
        {
            ForwardBack = -1f;
            playerVelocity = transform.forward * maxSpeed * ForwardBack;
        }
        else
        {
            ForwardBack = 0f;
            playerVelocity = transform.forward * maxSpeed * ForwardBack;
        }
        if (Input.GetKey(KeyCode.Keypad6))
        {
            RightLeft = 1f;
            playerVelocity += transform.right * maxSpeed * RightLeft;
        }
        else if (Input.GetKey(KeyCode.Keypad4))
        {
            RightLeft = -1f;
            playerVelocity += transform.right * maxSpeed * RightLeft;
        }
        else
        {
            RightLeft = 0f;
            playerVelocity += transform.right * maxSpeed * RightLeft;
        }

        if (axes == RotationAxes.MouseXAndY)
        {
            // Read the mouse input axis
            rotationX += Input.GetAxis("Mouse X") * sensitivityX;
            rotationY -= Input.GetAxis("Mouse Y") * sensitivityY;

            if (rotationY < -20f)
            {
                rotationY = -20f;
            }
            if (rotationY > 80f)
            {
                rotationY = 80f;
            }
            transform.RotateAround(transform.localPosition, Vector3.up,
rotationX - deltaX);
            deltaX = rotationX;
            if (MainCamera != null)
            {
                MainCamera.transform.RotateAround(transform.localPosition,
transform.right, rotationY - deltaY);
                deltaY = rotationY;
            }
        }
        controller.SimpleMove(playerVelocity);
    }
    void OnControllerColliderHit(ControllerColliderHit hit)
    {
        Rigidbody body = hit.collider.attachedRigidbody;
```

```
        if (body == null || body.isKinematic)
        {
            return;
        }
        if (hit.moveDirection.y < -0.3)
        {
            return;
        }
        Vector3 pushDir = new Vector3(hit.moveDirection.x, 0,
hit.moveDirection.z);
        body.velocity = pushDir * pushPower;
    }
    private void RunPorter()
    {
        if (porterAnim != null)
        {
            porterAnim.SetInteger("run", 1);
            porterAnim.SetInteger("walk", 0);
            porterAnim.SetInteger("idle", 0);
        }
    }
    private void WalkPorter()
    {
        if (porterAnim != null)
        {
            porterAnim.SetInteger("run", 0);
            porterAnim.SetInteger("walk", 1);
            porterAnim.SetInteger("idle", 0);
        }
    }
    private void IdlePorter()
    {
        if (porterAnim != null)
        {
            porterAnim.SetInteger("run", 0);
            porterAnim.SetInteger("walk", 0);
            porterAnim.SetInteger("idle", 1);
        }
    }
}
```

Приложение Б. Код поведения интеллектуального агента

Листинг 2 – Класс `PorterAgent`

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using static UnityEngine.GraphicsBuffer;
using System.Numerics;
using Unity.Mathematics;
using Unity.VisualScripting;
using UnityEngine.UIElements;
public class PorterAgent : Agent
{
    [SerializeField] private Transform boxTransform;
    [SerializeField] private Transform targetTransform;
    public Transform spawn01Transform;
    public Transform spawn02Transform;
    public override void OnEpisodeBegin()
    {
        transform.localPosition = new
UnityEngine.Vector3(UnityEngine.Random.Range(-5f, 4.5f), 1f,
UnityEngine.Random.Range(3.5f, 4f));
        if (UnityEngine.Random.Range(0, 2) >= 1)
        {
            boxTransform.localPosition = spawn01Transform.localPosition;
        }
        else
        {
            boxTransform.localPosition = spawn02Transform.localPosition;
        }
    }
    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(boxTransform.localPosition);
        sensor.AddObservation(targetTransform.localPosition);
    }
    public override void OnActionReceived(ActionBuffers actions)
    {
        float moveY = actions.ContinuousActions[0];
        float moveZ = actions.ContinuousActions[1];
        float speed = 1.5f;
        transform.Rotate(transform.up * moveY * speed, Time.fixedDeltaTime
* 200f);
        this.GetComponent<Rigidbody>().AddForce(transform.forward * speed *
moveZ, ForceMode.VelocityChange);
    }
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<float> continuousActions =
actionsOut.ContinuousActions;
        continuousActions[0] = Input.GetAxisRaw("Horizontal");
        continuousActions[1] = Input.GetAxisRaw("Vertical");
    }
}
```

Приложение В. Код управления временем и событиями

Листинг 3 – Класс `PorterAgent`

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class LVLTimer : MonoBehaviour
{
    public float timeRemaining = 150;
    public bool timerIsRunning = false;
    public TMP_Text _TimeText;
    private void Start()
    {
        timerIsRunning = true;
    }
    void Update()
    {
        if (Input.GetKey(KeyCode.Escape))
        {
            SceneManager.LoadScene("Menu");
        }
        if (Time.timeScale == 0)
        {
            _TimeText.text = "Пауза! Время: "
                             + _TimeText.text;
        }
        if ((Input.GetKey(KeyCode.P))
            && (Time.timeScale == 1))
        {
            Time.timeScale = 0;//пауза
        }
        else if ((Input.GetKey(KeyCode.P))
                 && (Time.timeScale == 0))
        {
            Time.timeScale = 1;//старт
        }
        if (timerIsRunning)
        {
            if (timeRemaining > 0)
            {
                timeRemaining -= Time.deltaTime;
                DisplayTime(timeRemaining);
            }
            else
            {
                _TimeText.text = "Время кончилось!";
                timeRemaining = 0;
                timerIsRunning = false;
            }        }
    }
    void DisplayTime(float timeToDisplay)
    {
        timeToDisplay += 1;
        float minutes = Mathf.FloorToInt(timeToDisplay / 60);
        float seconds = Mathf.FloorToInt(timeToDisplay % 60);
        _TimeText.text = string.Format("{0:00}:{1:00}",
                                       minutes, seconds);
    }
}
```