Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Кафедра информационных технологий и систем управления

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

Зав. кафедрой ИТиСУ
_____ Е.В. Кислицын
(подпись) (Ф.И.О.)
«04» 06 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА МОДЕЛИ ИНТЕЛЛЕКТУАЛЬНОГО АГЕНТА В ИГРОВОЙ СРЕДЕ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

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

Нормоконтролер: Бредихина Наталья Сергеевна _____

Студент группы: РИМ-220907 Косарев Максим Евгеньевич _____
подпись

Екатеринбург
2024

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ
Кафедра информационных технологий и систем управления
Направление подготовки 09.04.01 Информатика и вычислительная техника
Образовательная программа Инженерия искусственного интеллекта

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студента Косарева Максима Евгеньевича группы РИМ-220907
(фамилия, имя, отчество)

1. Тема выпускной квалификационной работы Разработка модели интеллектуального агента в игровой среде с использованием методов машинного обучения

Утверждена распоряжением по институту от «4» декабря 2023 г. № 33.02-05/298

2. Научный руководитель Денисов Дмитрий Вадимович, доцент, кандидат технических наук
(Ф.И.О., должность, ученая степень, ученое звание)

3. Исходные данные к работе Материалы, полученные в ходе преддипломной практики, техническая литература

4. Перечень демонстрационных материалов презентация в MS PowerPoint

5. Календарный план

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	Глава 1. Обзор методов создания моделей машинного обучения	до 23.03.2024 г.	Выполнено
2.	Глава 2. Разработка игровой среды и интеллектуальной модели агент	до 29.04.2024 г.	Выполнено
3.	Глава 3. Тестирование агента на возможность адаптации	до 19.05.2024 г.	Выполнено
4.	ВКР в целом	до 20.05.2024 г.	Выполнено

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

Студент задание принял к исполнению 12.02.2024 (подпись)
дата

6. Допустить Косарева Максима Евгеньевича к защите выпускной квалификационной работы в экзаменационной комиссии

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

РЕФЕРАТ

Выпускная квалификационная работа бакалавра 55 с., 28 рис., 45 источников.

ИНТЕЛЛЕКТУАЛЬНЫЙ АГЕНТ, МАШИННОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, UNITY ML-AGENTS, ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ, ТОП-ДАУН ШУТЕР, ПОВЕДЕНЧЕСКОЕ КЛОНИРОВАНИЕ, ГЕНЕРАТИВНОЕ СОСТЯЗАТЕЛЬНОЕ ИМИТАЦИОННОЕ ОБУЧЕНИЕ.

Цель работы – разработка и тестирование модели интеллектуального агента для игровой среды с использованием методов машинного обучения, способного адаптироваться к новым условиям.

Объект исследования – модель интеллектуального агента в игровой среде.

Методы исследования: анализ, синтез, моделирование, машинное обучение, тестирование, наблюдение, сравнение, эксперимент.

Результаты работы: разработана и протестирована модель интеллектуального агента в прототипе игры в жанре top-down shooter. Проведены эксперименты, определена оптимальная модель для адаптации агента к изменяющимся условиям игровой среды.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная работа посвящена искусственному интеллекту и его роли в контексте игровой индустрии. При разработке игровых проектов команда разработчиков сталкивается с большим количеством сложностей. С развитием технологий появляются новые возможности, благодаря этому можно упростить некоторые процессы, например проверку новых механик или уровней. С помощью моделей искусственного интеллекта, способных адаптироваться к разнообразным сценариям игры и принимать решения на основе окружающей обстановки.

Одним из важных аспектов разработки игр является тестирование новых уровней и механик игрового процесса. Процесс тестирования нововведений лежит на плечах тестировщиков или игроков, при этом компаниям приходится выделять бюджет на проведение тестов и поиска проблемных моментов.

В большом количестве игр реализован искусственный интеллект для взаимодействия с игроком или выполнения роли соперника, и можно использовать такое решение для тестирования, но есть ряд проблем. При любом изменении механик придется писать новый код; при использовании основных принципов создания ИИ, будь то дерево решений или конечный автомат состояний, после внедрения новых механик придется перенастраивать состояния, параметры в них, их приоритеты и многое другое. Понадобится большое количество рабочего времени, чтобы привести все в порядок, и после всего этого тестирование с помощью ИИ не будет считаться релевантным.

В этой работе рассматривается идея использования моделей машинного обучения для создания интеллектуальных агентов, способных эффективно взаимодействовать с игровым окружением. Такое решение может стать необходимым инструментом для тестирования новых уровней на предмет сложности, баланса и других интересующих метрик. Такой подход позволяет обеспечить более глубокое и точное тестирование игровых сценариев, что в конечном итоге повышает качество продукта.

Поэтому в работе была поставлена цель – разработать модель интеллектуального агента в игровой среде с использованием методов машинного обучения, с возможностью адаптации к новым условиям, а именно: изменение локации, параметров персонажа и его экипировки, количества союзников и противников и т.д. Для достижения цели необходимо решить следующие задачи:

1) Разработать прототип игры в жанре top-down shooter,
2) Реализовать интеллектуального агента с использованием методов машинного обучения,
3) Протестировать агента на возможность адаптации к новым условиям в игровой среде.

Для решения поставленных задач в работе используется следующий стек технологий: игровой движок Unity3D, набор инструментов и библиотек для разработки и обучения искусственных интеллектов Unity ML-Agents, фреймворк для управления зависимостями и внедрения зависимостей Zenject, язык программирования C#, среда разработки “Visual Studio” и система контроля версий Git.

Объектом исследования является интеллектуальная модель агента в игровой среде, созданная с использованием методов машинного обучения.

Предметом исследования являются методы машинного обучения, их применение и эффективность в контексте создания интеллектуальных агентов в игровой среде.

При написании данной работы использовались следующие методы: анализ, синтез, сравнение, абстрактно-логический и другие методы. В качестве теоретической и информационно-аналитической базы были использованы научные публикации российский и зарубежных авторов, информационные материалы, размещенные на сайтах сети интернет.

1 Обзор методов создания моделей машинного обучения

1.1 Обучение с подкреплением

В этой главе будет рассмотрено обучение с подкреплением. Это одна из разновидностей машинного обучения, представляет из себя подход, при котором агенты обучаются принимать решения на основе поощрений и наказаний. Этот подход широко используется в таких областях, как робототехника, игровая индустрия и других приложениях, где агентам необходимо научиться решать комплексные задачи методом проб и ошибок [5].

Добавлено примечание ([НБС1]): Оформление ссылок . В конце предложения перед знаком «.» Стр.56 методички. Исправить во всем тексте

Обучение с подкреплением происходит из ранних исследований в области психологии и изучения поведения животных. Основная идея заключается в том, что организм учиться принимать определенные решения, основываясь на последствии этих решений.

В 1930-х годах американский психолог Б.Ф. Скиннер ввел понятие оперантного обусловливания, которое представляет собой форму обучения, при которой поведение изменяется в зависимости от его результатов. Исследования Скиннера фокусировались на том, как можно обучить организмы реагировать на раздражители, опираясь на последствия их решений [6].

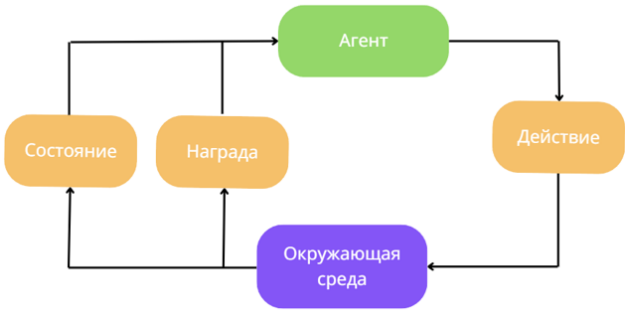


Рисунок 1 - Схема обучения с подкреплением

Рассмотрим, как работает алгоритм обучения с подкреплением. Алгоритм принимает различные решения, чтобы изучить соответствующие положительные и отрицательные ценности для достижения конечного результата – вознаграждения. Ознакомимся с основными понятиями:

- Агент – алгоритм машинного обучения;
- Среда - контекст, характеризующийся адаптивным пространством задач, которое содержит переменные, ограничения, правила и доступные действия;
- Действие – это шаг, который агент предпринимает для навигации по среде;
- Состояние – это среда в данный момент времени;
- Вознаграждение – это положительное, отрицательное или нулевое значение за выполнение действия;
- Совокупное вознаграждение – это сумма всех вознаграждений или конечное значение.

Обучение с подкреплением использует марковский процесс принятия решений, который основан на математическом моделировании принятия решений с использованием дискретных временных шагов. На каждом шаге агент выполняет новое действие, которое изменяет состояние окружающей среды. Текущее состояние также зависит от предыдущих действий агента [4].

В процессе перемещения по среде агент создает набор правил или политик в результате опыта и ошибок. Эти правила помогают агенту определить, какие действия предпринять, чтобы максимизировать общее вознаграждение. Кроме того, агент должен выбирать между исследованием окружающей среды в поисках новых вознаграждений и выбором известных действий с высокими вознаграждениями в текущем состоянии.

Не мало важно отметить преимущества использования алгоритмов обучения с подкреплением. Главное преимущество — это высокая эффективность в сложной среде с множеством правил, переменных и зависимостей. В такой ситуации человек не всегда способен принимать

правильные решения и выбирать наилучшую траекторию решений, даже обладая большими знаниями и опытом в данной среде. Вместо этого алгоритмы быстро адаптируются к постоянно меняющимся средам и находят новые стратегии для оптимизации результатов.

В стандартных алгоритмах машинного обучения при изменении среды необходимо вносить правки и обрабатывать новые данные, чтобы управлять алгоритмом и получать необходимый результат. При использовании обучения с подкреплением нет надобности человеческого вмешательство в структуру программы, так как алгоритм сам адаптируется к изменению среды.

Так как алгоритм ориентирован на максимизацию совокупного вознаграждения, он особо эффективен для сценариев, в которых действия влекут за собой длительные последствия. Особенно хорошо этот алгоритм показывает себя в ситуациях, где обратная связь по каждому шагу недоступна, поскольку может оцениваться правильность принятых решений на основе отложенных вознаграждений.

1.2 Unity Machine Learning Agents Toolkit

Unity ML-Agents (Machine Learning Agents) — это фреймворк, разработанный Unity Technologies, который предоставляет средства для обучения искусственного интеллекта (ИИ) в среде Unity. Он предоставляет разработчикам инструменты для создания сред с взаимодействующими агентами, которые могут обучаться с использованием различных методов машинного обучения, включая обучение с подкреплением, обучение с учителем и обучение без учителя [1]. Unity ML-Agents позволяет разработчикам создавать разнообразные приложения, такие как игры, симуляции, виртуальные тренировки и другие, в которых агенты могут обучаться и совершенствовать свои навыки на основе опыта. Unity ML-Agents состоит из пяти высокоуровневых компонентов.

Среда обучения - содержит сцену Unity и всех игровых персонажей. Сцена Unity обеспечивает среду, в которой агенты наблюдают, действуют и учатся. То, как будет настроена сцена Unity в качестве среды обучения, зависит от цели разработки. Она подходит для решения конкретной задачи обучения с подкреплением ограниченного масштаба, в этом случае можно использовать одну и ту же сцену как для обучения, так и для тестирования обученных агентов. Также есть возможность обучать агентов действовать в сложной игре или симуляторе. В этом случае, предоставляется возможность создать специальную сцену для тренировок.

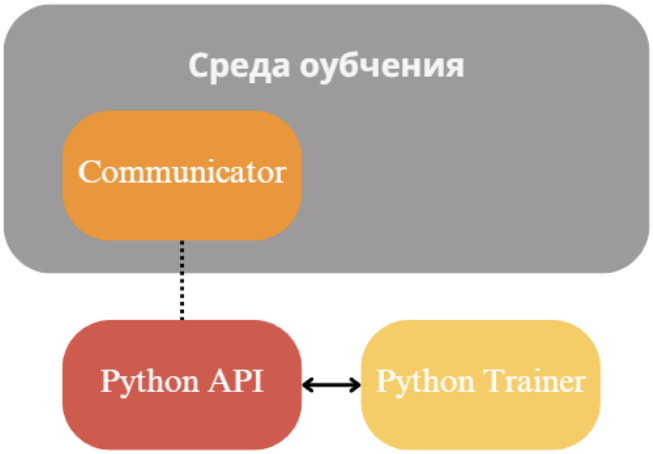
API Python - который содержит низкоуровневый интерфейс Python для взаимодействия со средой обучения и манипулирования ею. API Python не является частью Unity, а находится снаружи и взаимодействует с Unity через коммуникатор, используется в процессе обучения Python для связи с академией и управления ею во время обучения.

Внешний коммуникатор - который соединяет среду обучения с низкоуровневым API Python. Он находится внутри среды обучения.

Python Trainers - который содержит все алгоритмы машинного обучения, позволяющие обучать агентов. Алгоритмы реализованы на Python и являются частью пакета mlagents Python. Тренажеры Python взаимодействуют исключительно с низкоуровневым API Python.

Оболочка Gym - распространенным способом взаимодействия исследователей машинного обучения со средами моделирования является оболочка, предоставляемая OpenAI под названием gym.

Оболочка PettingZoo - это python API для взаимодействия со средами многоагентного моделирования, который предоставляет интерфейс, похожий на gym.



Добавлено примечание ([НБС2]): Оформление рисунков. Стр 53 методички. Исправить во всей работ убрать пустую строку между рисунком и подписью

Рисунок 2 - Упрощенная структурная схема агентов машинного обучения

Обучающая среда содержит два компонента Unity, которые помогают организовать сцену:

Агенты - которые прикрепляются к игровому объекту Unity и обрабатывают генерирование его наблюдений, выполнение действий, которые он получает, и назначение вознаграждения (положительного / отрицательного), когда это уместно. Каждый агент связан с определенным поведением.

Поведение (Behavior) - определяет конкретные атрибуты агента, такие как количество действий, которые может предпринять агент. Каждое поведение уникально идентифицируется полем Behavior Name. Поведение можно рассматривать как функцию, которая получает наблюдения и вознаграждения от Агента и возвращает действия. Поведение может быть одного из трех типов: обучающее, эвристическое или логический вывод.

Обучающее поведение — это поведение, которое еще не определено, но вот-вот будет обучено.

Эвристическое поведение — это поведение, которое определяется жестко заданным набором правил, реализованных в коде.

Логический вывод — это поведение, которое включает в себя обученный файл нейронной сети. По сути, после того, как обучающее поведение обучено, оно становится логическим выводом.

В каждой обучающей среде всегда будет по одному агенту для каждого персонажа в сцене. Хотя каждый агент должен быть связан с определенным поведением, возможно, что агенты, имеющие схожие наблюдения и действия, будут вести себя одинаково.

1.3 Глубокое обучение с подкреплением

В этой главе будут рассмотрены методы глубокого обучения с подкреплением, сигналы вознаграждения, блок любопытства с ограниченным вознаграждением, имитационное обучение, генеративное состязательное имитационное обучение и поведенческое клонирование.

1.3.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) является одним из наиболее распространенных методов обучения с подкреплением, который часто используется в качестве основного алгоритма, включая его применение в алгоритмах исследования среды агентом. PPO относится к группе методов, которые стремятся оптимизировать целевую функцию, оценивая ожидаемое значение награды, и является методом "on-policy", что означает, что обновления параметров происходят только на основе данных, собранных текущей политикой [9,14].

Одним из ключевых понятий, используемых в PPO, является траектория. В контексте обучения с подкреплением траектория представляет собой последовательность состояний среды, последовательно принимаемых

агентом. Начиная с начального состояния s0, действия агента выбираются в соответствии с текущей политикой, приводя к следующему состоянию s1, и так далее.

PPO разработан как альтернатива методу Trust Region Policy Optimization (TRPO), но с использованием методов оптимизации более низкого порядка. Основная идея заключается в том, чтобы максимально улучшить политику, используя небольшие траектории, чтобы избежать потери производительности. Обновления политики в PPO происходят путем максимизации функции, которая оценивает разницу между новой и старой политиками, с учетом ограничений, чтобы изменения были незначительными [15].

1.3.2 Soft Actor-Critic (SAC)

Идеальный алгоритм глубокого RL для систем реального мира должен обладать определенными характеристиками для решения таких задач, как перебои в потоках данных, вывод с низкой задержкой и плавное исследование, чтобы предотвратить механический износ роботов. Ключевые свойства алгоритма включают эффективность выборки, нечувствительность к гиперпараметрам и возможность обучения вне политики. Soft actor-critic (SAC) отвечает этим требованиям как алгоритм глубокого RL, не зависящий от политики и не зависящий от модели, поскольку он эффективен для выборки, устойчив к гиперпараметрам и применим в различных средах [2].

Эксперименты в реальном мире также требуют специальных функций реализации, включая асинхронную выборку, поддержку обучения остановке/возобновлению и сглаживание действий для предотвращения повреждения оборудования из-за высокочастотного дрожания привода. Soft actor-critic, основанный на платформе обучения с усилением максимальной энтропии, оптимизирует целевую функцию с увеличением энтропии, чтобы сбалансировать разведку и эксплуатацию. Эта цель может быть

интерпретирована как максимизация ожидаемой отдачи при максимизации энтропии политики, контролируемой параметром температуры. SAC достигает этого, параметризуя гауссову политику и Q-функцию с помощью нейронных сетей, оптимизируя их с помощью приближенного динамического программирования. Этот подход не только решает проблемы реального мира, но и обеспечивает самую современную производительность в моделируемых тестах без регуляризации энтропии.

1.4 Сигналы вознаграждения в обучении с подкреплением

Сигналы вознаграждения являются фундаментальными в обучении с подкреплением, направляя агентов к достижению желаемых целей. Два основных типа вознаграждений - внутренние и внешние, каждое из которых служит различным целям в процессе обучения [12].

При обучении с подкреплением агенты стремятся максимизировать совокупное вознаграждение, находя оптимальные стратегии. Внешние вознаграждения, определяемые окружающей средой, непосредственно соответствуют достижению конкретных целей и служат прямыми показателями успеха задачи.

В дополнение к внешним вознаграждениям могут быть введены внутренние сигналы вознаграждения для дальнейшего формирования поведения агента. Эти вознаграждения, независимы от окружающей среды, предназначены для поощрения конкретных действий или помощи в изучении истинного внешнего вознаграждения.

Инструментарий ML-Agents предлагает модульную структуру для определения сигналов вознаграждения, позволяющую гибко формировать поведение агентов. В инструментарии доступны четыре основных сигнала вознаграждения:

Extrinsic: представляет вознаграждения, определенные в среде, и включен по умолчанию.

Gail: внутренний сигнал вознаграждения, определяемый алгоритмом генеративного состязательного имитационного обучения (GAIL).

Curiosity: внутренний сигнал вознаграждения, способствующий исследованию в среде с ограниченным вознаграждением, реализуемый с помощью модуля Curiosity.

RND: Внутренний сигнал вознаграждения, поощряющий исследование в среде с ограниченным вознаграждением, определяемый модулем Random Network Distillation

Используя эту модульную структуру, исследователи и практики могут разрабатывать структуры вознаграждения, адаптированные к целям задач обучения с подкреплением, способствуя более эффективному обучению агентов [11].

1.5 Имитационное обучение

Имитационное обучение предлагает интуитивный подход к обучению агентов путем прямой демонстрации желаемого поведения, а не полагаться исключительно на методы проб и ошибок. Например, при обучении агента игровому сценарию вместо определения сложной функции вознаграждения мы можем предоставить реальные примеры наблюдений и соответствующих действий с игрового контроллера. Посредством имитационного обучения агент усваивает политику, основанную на парах наблюдений и действиях на демонстрациях, что способствует более эффективному процессу обучения [8].

Имитационное обучение может применяться как самостоятельно, так и в сочетании с обучением с подкреплением. При самостоятельном использовании оно позволяет агенту научиться определенному поведению или стилю решения задач. В сочетании с обучением с подкреплением оно значительно ускоряет процесс обучения, особенно в условиях скудного вознаграждения. Ниже предоставлен график обучения агентов в среде одного из демонстрационных проектов Unity ML-Agents, с применением

имитационного обучения. На графике ниже мы можем увидеть четырехкратную эффективность данного метода обучения.

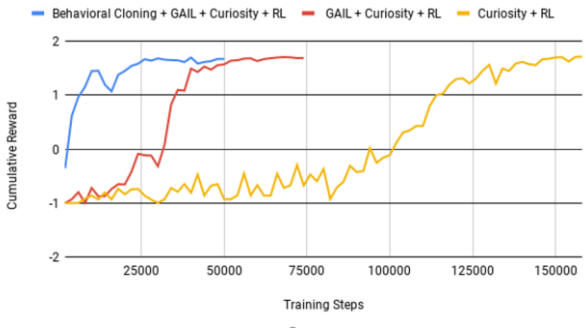


Рисунок 3 – Сравнение различных подходов обучения с подкреплением

Инструментарий ML-Agents предлагает механизмы для непосредственного включения демонстраций в обучение и ускорения процессов обучения, основанных на вознаграждении. Для поддержки этой интеграции предусмотрены два алгоритма: поведенческое клонирование (BC) и генеративное состязательное имитационное обучение (GAIL). В большинстве сценариев эти функции могут быть объединены для оптимизации результатов обучения.

1.5.1 Генеративное состязательное имитационное обучение (GAIL)

Генеративное состязательное имитационное обучение, обычно сокращаемое как GAIL, представляет собой инновационный подход к обучению агентов путем использования методов состязательности для поощрения поведения, сходного с набором демонстраций [17]. Эта методология демонстрирует эффективность как в сценариях с вознаграждениями среды, так и без них, особенно когда ограничено количество записанных демонстраций.

В рамках GAIL вводится вторичная нейронная сеть, называемая дискриминатором. Задача дискриминатора различать наблюдения и действия, полученных на основе демонстраций, и те, которые генерируются агентом. Посредством итеративного обучения дискриминатор совершенствует свою способность различать наблюдения и действия, тем самым подавая сигнал вознаграждения, основанный на сходстве между поведением агента и предоставленными демонстрациями.

На каждом этапе обучения агент стремится максимизировать сигнал вознаграждения, подаваемый дискриминатором. Одновременно дискриминатор проходит дальнейшее обучение для улучшения своих возможностей распознавания, постоянно повышая планку производительности агента. Следовательно, по мере того, как агент постепенно улучшает свою способность подражать демонстрируемому поведению, дискриминатор становится более проницательным, что требует от агента все больших усилий по его обману.

Эта состязательная структура позволяет изучать политику, которая приводит к состояниям и действиям, очень похожим на те, которые наблюдались во время демонстраций, часто требуя меньшего количества демонстраций, чем другой подход, который будет рассмотрен в следующей главе, поведенческое клонирование. Более того, в дополнение к обучению исключительно на демонстрациях, сигнал вознаграждения GAIL может быть легко интегрирован с внешним сигналом вознаграждения, обеспечивая дополнительное руководство процессом обучения и еще больше повышая эффективность обучения.

1.5.2 Поведенческое клонирование (BC)

Поведенческое клонирование (BC) представляет собой фундаментальный подход в обучении агентов, направленный на точное воспроизведение действий, продемонстрированных в заранее определенном

наборе примеров [3]. Эта функция легко интегрируется в тренажеры Proximal Policy Optimization (PPO) или Soft Actor-Critic (SAC), предлагая универсальность в реализации. Однако важно отметить, что BC не обладает способностью обобщать за пределами примеров, представленных в демонстрациях.

Поведенческое клонирование демонстрирует оптимальную производительность, когда полный набор демонстраций охватывает почти все возможные состояния, с которыми может столкнуться агент [16]. Альтернативно, BC можно эффективно комбинировать с генеративным состязательным имитационным обучением (GAIL) и/или внешним сигналом вознаграждения для расширения его возможностей. Интегрируя BC в более широкую систему обучения, агенты могут извлечь выгоду из улучшенных результатов обучения и улучшенной адаптивности в различных средах.

1.6 Внедрение зависимостей

Для создания комплексной игровой среды способной содержать в себе логику игры и обучать агентов, при этом поделив сцену на комнаты для параллельного обучения агентов, необходимо выстроить гибкую расширяемую архитектуру. Для этого в проекте используется внедрение зависимостей, реализованное с помощью Zenject.

Внедрение зависимостей — это шаблон проектирования программного обеспечения, широко используемый при разработке сложных систем, облегчающий управление зависимостями между различными компонентами [19]. В контексте Unity, популярного игрового движка и платформы разработки, DI играет решающую роль в повышении модульности, тестируемости и сопровождаемости проектов. В этой главе рассматриваются концепции внедрения зависимостей и его применение в Unity, опираясь на соответствующую литературу и ресурсы.

Внедрение зависимостей основано на принципе разделения компонентов путем делегирования ответственности за разрешение зависимостей внешнему объекту. Вместо того, чтобы компоненты напрямую создавали свои зависимости, они получают их из внешнего источника, обычно из инжектора зависимостей. Такой подход способствует гибкости, масштабируемости и простоте обслуживания за счет уменьшения тесной связи между компонентами.

Внедрение зависимостей может быть реализовано с использованием различных методов, таких как внедрение конструктора, внедрение свойств или внедрение метода. Каждый метод обладает определенными преимуществами и подходит для различных сценариев, основанных на требованиях проекта и конструктивных соображениях.

При разработке на Unity оно играет важную роль в управлении зависимостями между игровыми объектами, скриптами и другими компонентами. Для Unity доступно несколько фреймворков и библиотек DI, каждая из которых предлагает уникальные функции и возможности. Известные примеры включают Zenject, встроенную адресуемую систему Unity, и внешние библиотеки, такие как Extenject и StrangeIOC [20]. Эти фреймворки предоставляют утилиты для привязки зависимостей, внедрения и разрешения, позволяя разработчикам внедрять лучшие практики в области проектирования программного обеспечения и архитектуры.

При этом важно учитывать, что, использование DI в сочетании с другими архитектурными шаблонами, такими как Model-View-Controller (MVC), Model-View-Presenter (MVP) или Model-View-ViewModel (MVVM), может улучшить организацию кода и разделение задач.

1.7 Архитектурные шаблоны

Архитектурные шаблоны Model-View-Controller (MVC), Model-View-Presenter (MVP) и Model-View-ViewModel (MVVM) обеспечивают

структурированные подходы к разработке программного обеспечения, облегчая разделение задач и повышая удобство сопровождения кода. В этой главе рассматривается применение этих архитектурных шаблонов в разработке игр Unity, опираясь на информацию из соответствующей литературы и ресурсов.

Архитектурные шаблоны, такие как MVC, MVP и MVVM, направлены на отделение уровня представления от уровней бизнес-логики и доступа к данным, обеспечивая модульность и облегчая повторное использование кода. Каждый шаблон определяет отдельные роли и обязанности компонентов в архитектуре программного обеспечения.

1.7.1 MODEL-VIEW-CONTROLLER

Model-View-Controller (MVC) [18]: MVC делит приложение на три взаимосвязанных компонента:

Модель: представляет данные приложения и бизнес-логику.

Представление: отображает данные пользователю и фиксирует вводимые пользователем данные.

Контроллер: является посредником между моделью и представлением, обрабатывает вводимые пользователем данные и соответствующим образом обновляет модель.

Плюсы MVC:

- Разделение ответственности,
- Повторное использование кода,
- Тестируемость.

Минусы MVC:

- Слишком большая многослойность,
- Сцепление между слоями,
- Сложность тестирования некоторых сценариев,

В целом, MVC является мощным и гибким шаблоном проектирования, который может быть полезен для разработки различных приложений. Однако важно понимать его плюсы и минусы, прежде чем решать, использовать ли его для своего проекта.

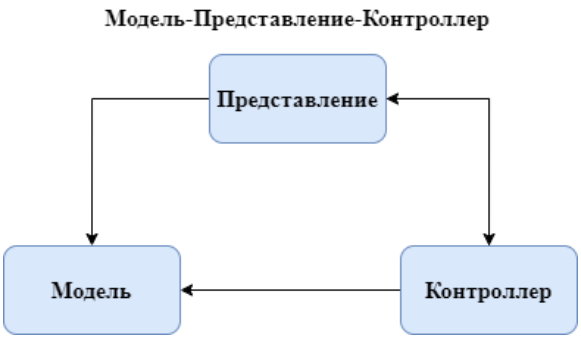


Рисунок 4 – Схема Model-View-Controller

1.7.2 MODEL-VIEW-PRESENTER

Model-View-Presenter (MVP): MVP повышает тестируемость, отделяя представление от бизнес-логики. Он состоит из трех компонентов:

Модель: представляет данные приложения и бизнес-логику.

Представление: отображает данные пользователю и пересылает пользовательский ввод представителю.

Представитель: является посредником между моделью и представлением, обрабатывает пользовательский ввод и обновляет модель.

Плюсы MVP:

- Тестируемость,
- Разделение ответственности,
- Повторное использование кода,
- Слабосвязный код.

Минусы MVP:

- Сложность принципа и реализации,
- Избыточность кода,
- Не подходит для простых приложений.

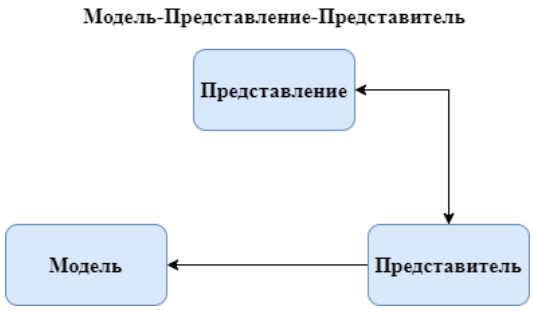


Рисунок 5 – Схема Model-View-Controller

1.7.3 Model-View-ViewModel

Model-View-ViewModel (MVVM): MVVM вводит ViewModel для управления логикой представления, отделяя ее от представления. Она включает в себя:

Модель: представляет данные приложения и бизнес-логику.

Представление: отображает данные пользователю и перенаправляет пользовательский ввод в ViewModel.

ViewModel: предоставляет данные и команды представлению, облегчая привязку данных и обновление модели.

Плюсы MVVM:

- Разделение ответственности (улучшенное),
- Улучшенная связь,
- Повторное использование кода.

Минусы MVVM:

- Повышенная сложность,

- Высокий уровень абстракции,
- Не подходит для всех случаев.

При разработке игр на Unity эти архитектурные шаблоны дают такие преимущества, как улучшенная организация кода, возможность тестирования и сопровождения.

Интеграция MVC, MVP или MVVM в проекты Unity предполагает структурирование кодовой базы в соответствии с принципами каждого шаблона. Это включает в себя отделение логики представления от бизнес-логики, использование методов привязки данных и реализацию взаимодействия между компонентами.

1.8 Примеры проектов с использованием Unity ML-Agents

Фреймворк разработан более 5 лет назад, несмотря на это в открытом доступе достаточно затруднительно найти готовые проекты. Unity предоставляет обучающие проекты, но они достаточно простые по механике и скорее это демонстрация возможностей, чем готовый проект.

Рассмотрим три проекта, которые часто обсуждаются и всплывают при изучении темы. Первый из них это парковка машины. На изображении ниже можно увидеть, игровую среду и агента в роли машины. Основная цель проехать от точки появления до парковочного места. Проект получается достаточно простым, а механика неприменима для игровых проектов или решения задач автопилота в реальном мире.

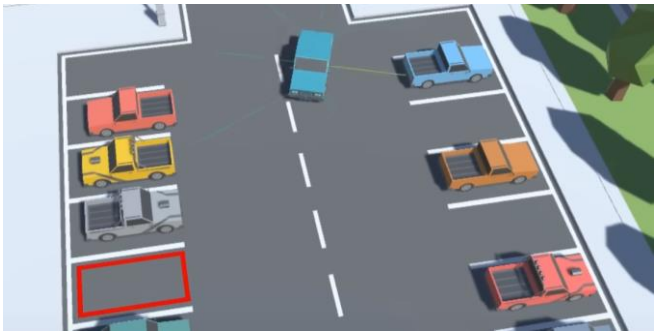


Рисунок 6 – Проект по парковке машины

Второй проект это роботическая рука, Unity предоставляют доступ к коду проекта. В нем рука переносит объекты с одного места на другое в разных условиях. На основе этого проекта сделаны реальные прототипы роборук, которые используются на предприятиях, что является отличным результатом, но не подходит под тему данной работы.

Рисунок 7 – Проект роборука

Третий проект разработала компания OpenAI. Проект называется – “Multy-Agent Hide-and-seek”. Агенты делятся на две команды, одни прячутся другие ищут. Есть игровая среда, на которой расположены преграды, есть три вида преград: статичные, активные и рампы. Агенты могут передвигать активные преграды и рампы и фиксировать их положение. Рампы позволят преодолевать преграды, при правильном их расположении.

Проект использует и комбинирует большое число механик, заточенных на передвижение агентов. Разнообразие игровых объектов создает огромное количество игровых ситуаций. При этом проект не подразумевает настройку баланса, так как не имеется изменяемых переменных, будь то скорость передвижения или класс агента, который может взаимодействовать только с определенными объектами.

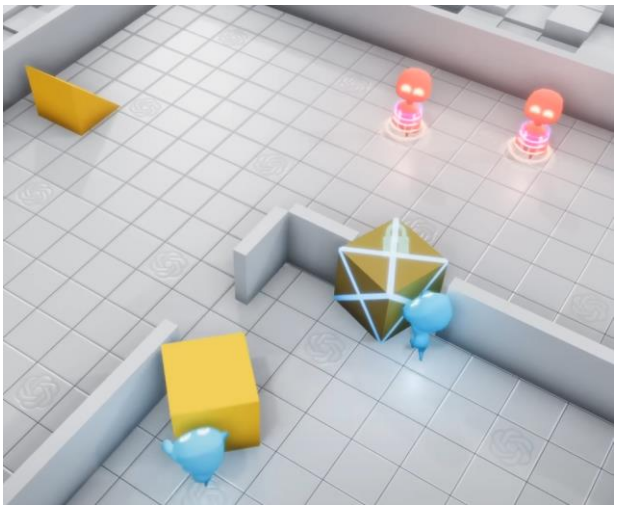


Рисунок 8 – Проект прятки

По представленным выше проектам видно, что с помощью данного инструмента создаются проекты разной направленности и он предоставляет достаточно широкий набор методов для решения поставленных задач.

2 Разработка игровой среды и интеллектуальной модели агента

2.1 Описание игровой среды

В качестве игровой среды, была придумана и разработана 3D игра в жанре top-down shooter, с помощью игрового движка Unity. Основной геймплей заключается в том, что пользователь управляет персонажем, передвигается по игровой среде, уничтожает противников с помощью оружия и увеличивает свой отряд, находя союзных юнитов на арене. Условия победы – это уничтожения всех соперников, условия поражения – это гибель юнита игрока или истечения таймера.

Действия происходят на закрытой плоской арене, на которой присутствуют препятствия, в виде мешей с коллайдерами различных форм и размеров — это оранжевые объекты и границы уровня – это красные объекты. Границы и препятствия имеют одинаковую логику как для обучения модели интеллектуального агента, так и для снарядов и юнитов. Одна такая арена представляет из себя уровень игры.

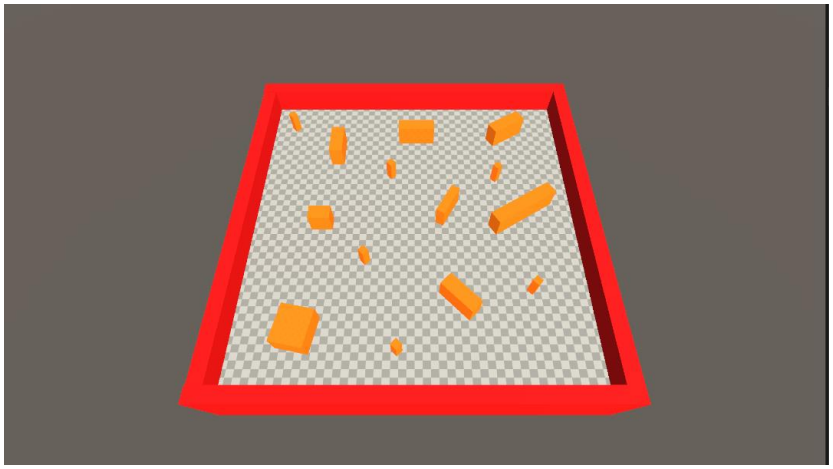


Рисунок 9 – Игровая среда

2.1.1 Система юнитов

При старте уровня на арене создаются противники, союзники и сам юнит пользователя, они же юниты. Юнит пользователя и союзные юнити появляются сразу. Противники появляются с настраиваемой задержкой в течении уровня. Для юнитов была создана система, благодаря которой, можно создать большое количество разных классов. Класс собирается из нескольких блоков: здоровье, передвижение, экипировка и визуальное отображение.

Блок здоровья состоит из двух основных параметров: максимальное количество здоровья и восстановление здоровья. При настройке восстановления здоровья есть возможность менять значения параметров: временной задержки между последним получением урона и началом восстановления, а также количество восстановления за тик и временные промежутки между тиками. Также, есть возможность отключения функции восстановления здоровья.

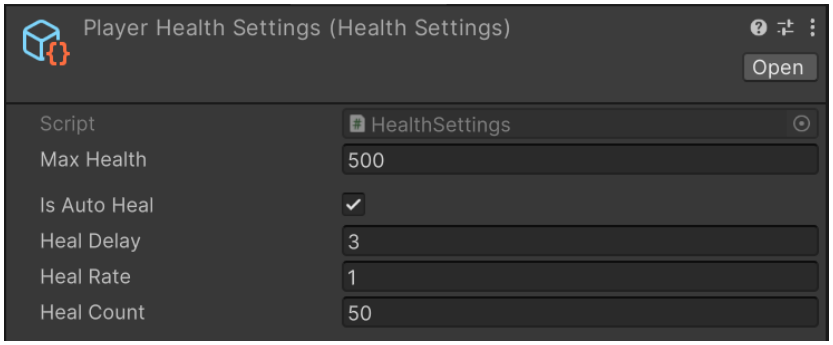


Рисунок 10 – Настройки блока здоровья

Блок передвижение состоит из трех параметров: скорость передвижения, скорость поворота и опционально максимальная дистанция преследования. Максимальная дистанция используется вражескими юнитами, при отдалении юнита игрока на дистанцию большею, чем это значение вражеский юнит останавливается до тех пор, пока пользователь не войдет в зону его действия.

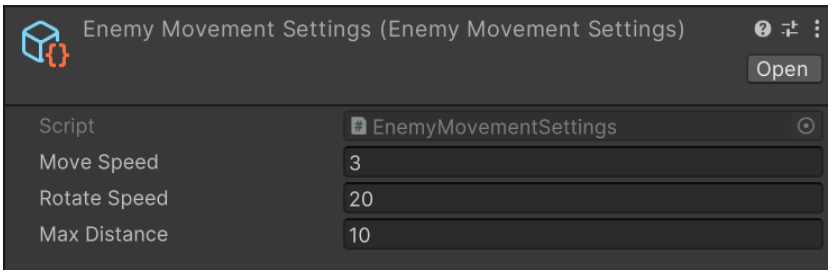


Рисунок 11 – Настройки блока передвижения

Также есть возможность настраивать визуальное отображение юнита для различия типов противников и союзников и их экипировки.

2.1.2 Система экипировки

Система экипировки устроена сложнее, чем предыдущие. Снаряжение юнитов — это оружие. Оно может быть разным по типу действия. Первая вариация представляет из себя ближний удар вокруг юнита, где можно настроить радиус действия и время задержки между ударами. Второй тип более интересен — это огнестрельное оружие, которое выпускает снаряды. При этом есть возможность выбрать паттерн выпускания снарядов например: выстрел одиночного снаряда по прямой – пистолет, выстрел по прямой нескольких пуль – автомат, выстрел по конусу нескольких снарядов – дробовик.

Также настраиваются слои коллизий пули, и слои, которые препятствуют прицеливанию. При работе оружия учитывается есть ли преграда между юнитом и его целью, и в случае ее отсутствия производится атака.

Помимо паттерна атаки у оружия есть время перезарядки, в которое не производятся выстрелы. Также у оружия есть визуальный эффект, который появляется при коллизии снаряда с препятствием или юнитом противоположной команды.

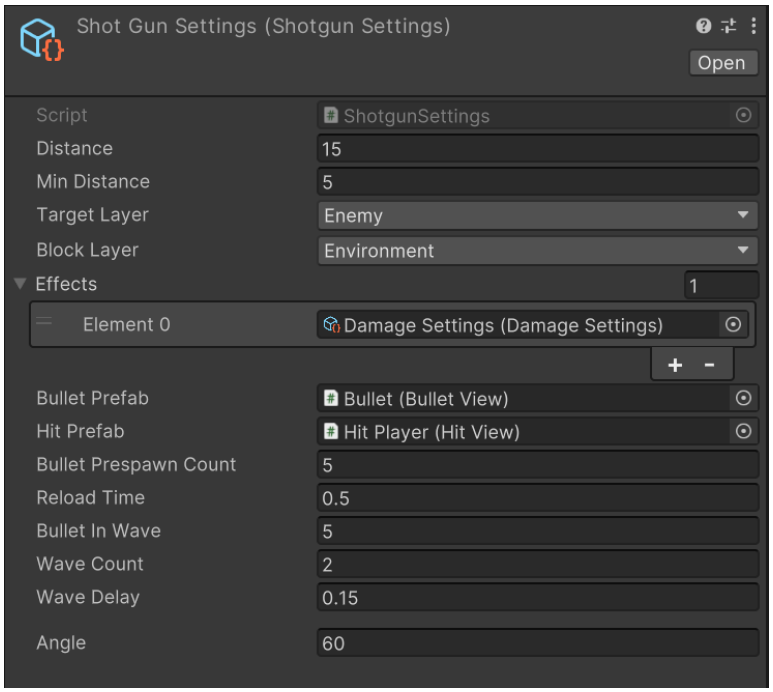


Рисунок 12 – Настройки блока экипировка

Для оружия можно определить тип снаряда\пули. Снаряды отличаются скоростью полета, траекторией полета и конечным действием например: применением эффектов на соперника или при столкновении с поверхностью нанесение эффектов по области – взрыв, пронизывающий полет, при котором пуля не исчезает после первого столкновения с юнитов соперника.

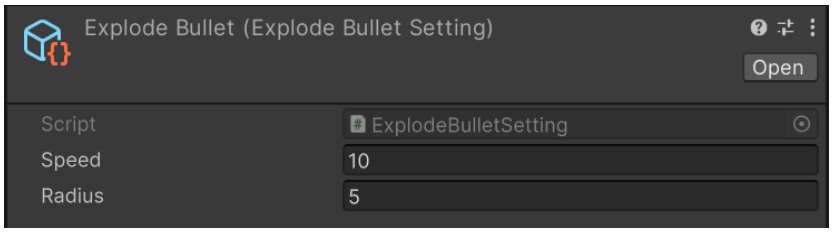


Рисунок 13 – Настройки снаряда

Также разработана система эффектов, при настройке экипировки указывается список эффектов, которые будут применены к цели при ее поражении, от базового нанесения урона, до замедления и тикающего урона, не позволяющего работать восстановлению здоровья.

Из полученной системы можно собирать уровни игровой среды с разной сложностью и длительностью прохождения. Благодаря, комбинации различных типов юнитов и их количеству как союзников, так и противников, и возможностью контролировать частоту появления новых вражеских юнитов, получилась игровая среда, в которой можно ставить эксперименты по обучению системы машинного обучения с подкреплением в постоянно изменяющейся среде.

2.1.3 Интерфейс

Важной частью любой игры является пользовательский интерфейс. В игре показывается количество оставшихся врагов и таймер, с количеством оставшегося времени. При использовании системы комнат, которая будет рассмотрена в следующей главе, интерфейс первой комнаты остается включенным, а все остальные выключаются, для возможности наблюдения за процессом обучения модели агента.

2.1.4 Система комнат

Для решения задачи машинного обучения всего одной игровой среды будет мало, так как обучение будет занимать большое количество времени. В пакете Unity ML-Agents есть поддержка одновременного обучения нескольких игровых сред. При запуске обучения мы можем использовать несколько комнат, количество которых ограничено только техническими возможностями устройства, используемого для обучения. Плюсом в игровом движке Unity есть возможность изменения масштаба, по которому проходит время

(ускорение игрового процесса). Совмещая две техники, получается ускорить процесс в несколько раз, что крайне важно на начальном этапе, когда необходимо подбирать комбинации параметров агента и модели машинного обучения, о которых пойдет речь в следующих главах.

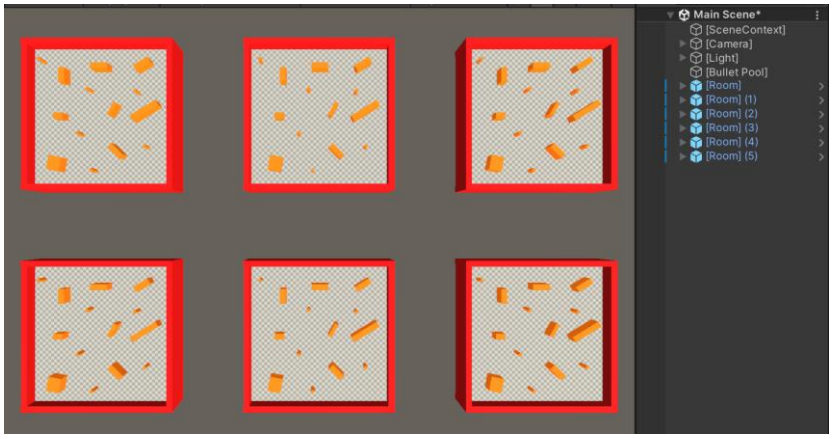


Рисунок 14 – Реализация системы комнат

Поскольку сначала была разработана игровая среда, а после в нее интегрировался пакет по обучению интеллектуальной модели игрока, в архитектуру игры пришлось вносить изменения. Благодаря использованию внедрения зависимостей, реализованное с помощью Zenject, не потребовалось больших усилий для перехода на систему комнат.

Все что было описано ранее соединялось по средствам SceneContext, там создавались фабрики для всех видов юнитов: игрок, союзники, враги. Определялись все сигналы, для передачи событий, подключался интерфейс и часть, которая следит за прогрессом прохождения уровня. Все что нужно сделать при такой системе, это перенести содержимое метода InstallBindings в скрипт инсталлера комнаты, который является GameObjectContext, а в SceneContext биндить сами комнаты. И второй момент — это перезагрузка комнаты при победе или поражении. На этот случай основные компоненты,

перечисленные в начале, наследуют интерфейс сброса, при активации которого комната откатывается до начального состояния и происходит новая инициализация.

Этот момент достаточно важен, для понимания, так как при попытке использования машинного обучения в игровых проектах, перенос игрового процесса по комнатам главная проблема, с которой столкнутся разработчики и при правильном подходе к изначальному выбору стека технологий и архитектуре процесс перехода не займет много времени.

Таким образом, получается система, при которой одновременно можно запускать несколько комнат, которые работают независимо друг от друга, тем самым кратно сокращая время обучения модели.

2.2 Разработка агента

Для разработки интеллектуальной модели агента в игровой среде с помощью методов машинного обучения необходимо определить параметры, по которым будет оцениваться эффективность модели. Ниже перечислим основные параметры, которые были сформулированы в ходе разработки модели игрока:

- Время прохождения уровня,
- Нанесение урона противникам,
- Получение урона,
- Активация союзных юнитов.

2.2.1 Основные методы

Для работы над моделью Unity ML-Agent предоставляет класс Agent, для наследования и дальнейшего расширения, и переопределения методов:

- OnActionReceived,
- CollectObservations,

- Heuristic.

Метод OnActionReceived выполняет действие, выбранное в соответствии с политикой агента, и назначает вознаграждение в соответствии с текущим состоянием. В этом методе принимается массив действий и преобразуется в направление движения юнита пользователя, через интерфейс сервиса ввода. И производится оценка действий агента.

```
/// <summary>
/// Вызывается каждый шаг движка. Здесь агент выполняет действие.
/// </summary>
Ссылок: 15
public override void OnActionReceived(ActionBuffers actionBuffers)
{
    //Установка направления движения юнита
    Vector3 direction = GetMoveDirection(actionBuffers.ContinuousActions);
    inputService.SetMoveDirection(direction);

    //Отрицательная оценка за каждый шаг
    var reward = GetReward(timeRewardCurve, timeReward);
    UpdateRewardValue(-reward);

    //Оценивание дистанции до соперников
    SetEnemiesDistanceReward();
}
```

Рисунок 15 – Определение метода OnActionReceived

Метод CollectObservations собирает данные наблюдения агента за окружающей средой. В ходе разработки и экспериментов с собиранием данных, был выбран подход, при котором указываются нормализованные данные координат. Как можно заметить на рисунке 15 первые два наблюдения — это координаты юнита пользователя. Далее собирается информация о ближайших соперниках, также их координаты. Стоит заметить, что собираются координаты осей X и Z, так как по оси Y значение всегда одинаковое. Далее собираются данные о ближайших союзниках, и после о ближайших снарядах соперника.

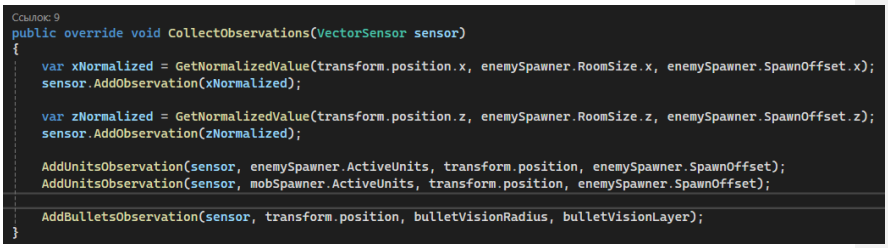


Рисунок 16 – Определение метода CollectObservations

Также, используется блок сенсорных лучей, для получения информации о преградах в игровой среде. Из центра агента выпускается 15 лучей на 360 градусов, длинной 10, которые взаимодействуют со слоем преград.

Метод Heuristic переопределяется для того, чтобы при разработке была возможность проверить игровую среду, не запуская при этом процесс обучения. Здесь используется управление с клавиатуры или джойстика на экране и записывается как действия агента. Контроль над действиями в ходе разработки помогает сохранить большое количество времени при проверке внедрения изменений в структуру агента или игровой среды.

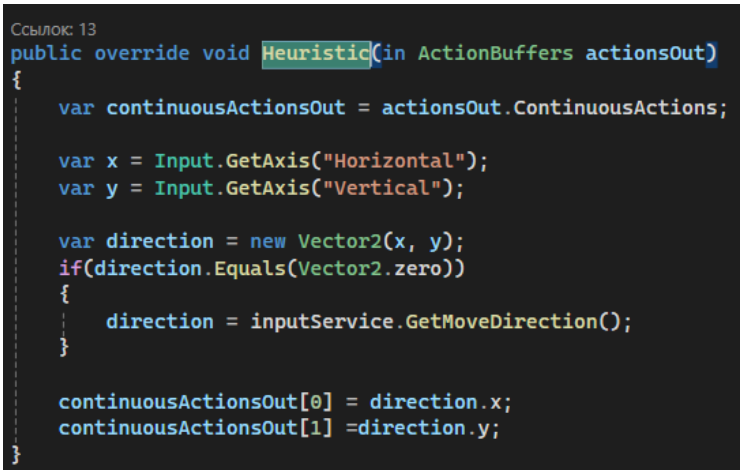


Рисунок 17 – Определение метода Heuristic

2.2.2 Награда агента

При оценивании эффективности принятых агентом решений надо смотреть на основные параметры, перечисленные выше. Из первого пункта следует, что агент должен стремиться проходить уровень за минимальный промежуток времени. Говоря об архитектуре работы класса агента, временным промежутком, стоит называть количество шагов, за которое уровень будет преодолен. Для стимуляции агента на использование меньшего количества шагов, на каждом шаге он получает отрицательную награду. Чем больше интеллектуальная модель агента затягивает прохождение уровня, тем ниже среднее совокупное вознаграждение.

Вторым параметром эффективности является нанесение урона, ведь уровень считается пройденным, когда все соперники повержены. Из этого следует логичный вывод, при нанесении урона начисляется положительная награда.

При использовании этого подхода, добиться хороших метрик и человеческого поведения агента крайне затруднительно. Проблема заключается в том, что награда начисляется в момент нанесения урона, а урон наносится посредством снаряда или области вокруг юнита, в зависимости от экипировки. За то время, что снаряд достигает своей цели, агент делает большое количество действий, которые на самом деле могут вести к противоположному эффекту или как минимум не тому, который нас интересует. Для решения этой проблемы и придания агенту более “человеческих” движений, в систему экипировки была добавлена минимальная и максимальная дистанция применения. Каждый шаг агента, измеряется дистанция до ближайшего юнита соперника, и в случае, если дистанция внутри необходимого промежутка, то начисляется положительная оценка, в ином случае отрицательная. В этом случае, награда дается непосредственно в момент, когда агент находится в правильной позиции.

Для оценки третьего параметра – получение урона, необходимо собрать информацию о снарядах, находящихся ближе всего к агенту и при этом, двигающихся в его сторону. В игровой среде производится поиск ближайших пуль и высчитывается дистанция до игрока, а также направление полета. При условии, что дистанция выбранных пуль от агента больше установленной, начисляется положительная оценка, в ином случае отрицательная.

При передвижении по игровой среде юнит может сталкиваться с коллайдерами и ходить вдоль границ уровня. При просмотре результатов обучения модели в игровой среде это первое, что бросается в глаза. Для того чтобы избежать подобного рода поведения, была добавлена отрицательная награда при столкновении с препятствиями и каждый шаг при нахождении юнита внутри коллайдера или преграды.

В качестве углубления и усложнения игровой механики, были добавлены союзные юниты, которые появляются в случайных местах и ожидают активации. Она происходит при приближении юнита пользователя на определенную минимальную дистанцию. После активации юниты присоединяются к отряду игрока и передвигаются за ним, используя экипировку, наносят урон по соперникам, тем самым помогают пройти уровень быстрее. В момент активации начисляется единовременная большая положительная награда.

Таким образом, за время прохождения уровня агент успевает сделать некоторое количество действий, за каждое из которых он получает оценку. Все это собирается в совокупное вознаграждение. По анализу динамики изменения и конечному значению совокупного вознаграждение делаются выводы о качестве обученной модели.

2.3 Обучение моделей

В предыдущих разделах были рассмотрены основы игровой среды, геймплей и механика обучения интеллектуальной модели агента. Теперь

настало время рассмотреть, как эти концепции реализуются в игровом движке, а также как они настраиваются.

Первым рассмотрим блок наград, агент получает ссылку на ScripableObject, в котором настраиваются параметры награды, которые обсуждались в предыдущей главе. Выбран именно такой метод реализации для возможности расширения функционала обучения и возможности корректировки значения в процессе проверки изменений игровой среды. Так же на рисунке ниже представлены AnimationCurve для каждой награды. С их помощью, можно увеличивать или уменьшать награду с течением времени с начала запуска или перезапуска уровня. Таким образом, можно достичь разного поведения обученной модели например: сделать акцент на быстрой активации союзников или наоборот полного их игнорирования.

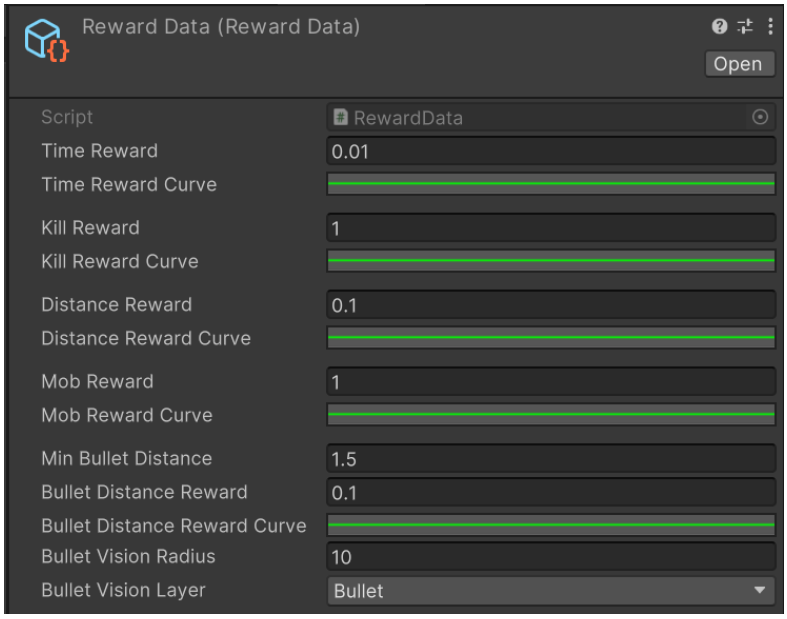


Рисунок 18 – Настройка блока наград

Настройка основных параметров модели. Имя агента может быть произвольным; оно указывается в конфигурации модели. Space Size равен

количеству собираемых наблюдений. Для обучения в первом эксперименте использовалось по одному значению нормализованной позиции юнитов внутри комнаты, а именно: позиции игрока, ближайшего противника, союзника и вражеского снаряда. Actions всего два – требуется только управление по осям X и Z, а это два числа с плавающей точкой

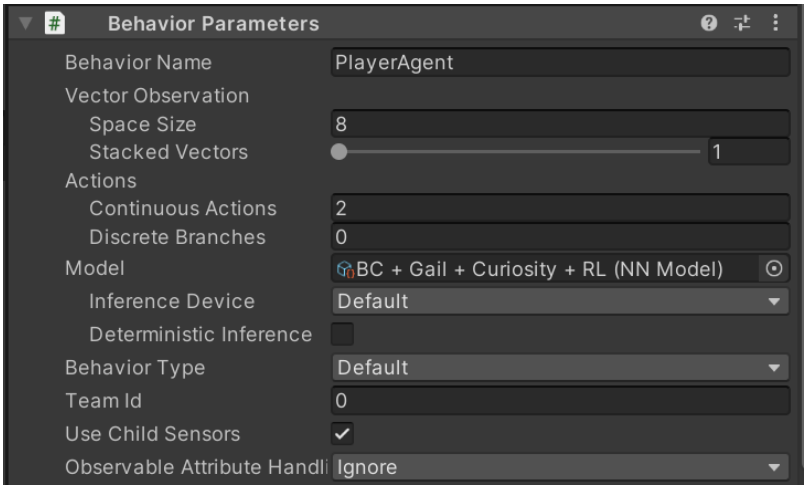


Рисунок 19 – Настройка основных параметров модели

Цикл наблюдения-принятия-решения-действия-вознаграждения повторяется каждый раз, когда агент запрашивает решение. Поскольку в геймплее нет симуляции физики, достаточно выбрать значение периода запроса решений, от этого показателя будет меняться итоговая оценка, что стоит учитывать при проведении экспериментов с изменением периода запроса решений. Экспериментальным путем было выбрано решение 10, то есть раз в 10 шагов будет запрошено решение. При значениях меньше агент начинает выполнять много мелких движений в разные стороны. При значениях больше агент не успевает уклоняться от снарядов и начинает промахиваться при необходимости повернуться.

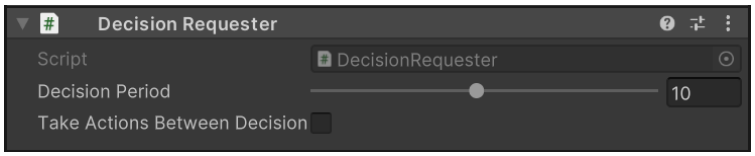


Рисунок 20 – Настройка компонента, запрашивающего принятие решения

Далее рассмотрим каких результатов получилось достигнуть при обучении моделей. Для этого будет проводиться эксперимент в игровой среде с препятствиями, показанными на рисунке 9. На арене будет 20 юнитов соперника и 5 союзных юнитов, при этом настройки юнитов будут одинаковыми. Сначала рассмотрим параметры модели, а после результаты эксперимента по 3 пунктам:

1. Совокупное вознаграждение,
2. Количество шагов для достижения желаемой оценки,
3. Поведение агента в игровой среде.

2.3.1 Базовая модель

На примере базовой модели рассмотрим основные параметры, которые будут одинаковыми во всех проводимых экспериментах. Таблица 1 включает в себя значения основных гиперпараметров модели.

Таблица 1 – Гиперпараметры модели

Поле	Значение
trainer_type	ppo
batch_size	128
buffer_size	2040
learning_rate	0.0003
beta	0.01

Добавлено примечание ([НБС3]): Оформление таблицы. Указать «продолжение таблицы» и указать номер таблицы. Название таблицы над таблицей, См стр 50 методички. Исправить во всей работе

Продолжение таблицы1

epsilon	0.3
lambd	0.95
num_epoch	3
learning_rate_schedule	linear
hidden_units	512
num_layers	2
vis_encode_type	simple

Модели, рассматриваемые в этой главе, будут отличаться подходом к оценке наград агента, для базовой модели в первом эксперименте используется блок внешней награды. На рисунке ниже представлены результаты обучения, первый график показывает значение совокупного вознаграждения, а второй среднюю продолжительность эпизода.

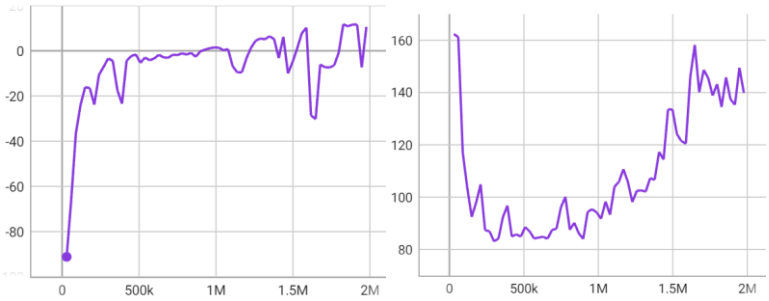


Рисунок 21 – Графики совокупного вознаграждения и продолжительности прохождения

Среднее значение прохождения уровня игроком, составляют 40 единиц совокупного вознаграждения и время прохождения эпизода 240 секунд. Модель обучалась на 1 980 000 шагах, это заняло 6 часов. Базовая модель достигает значений в 10 и 140. Поскольку на работу модели можно посмотреть

не только на графиках, но и непосредственно пронаблюдать за ее поведением в игровом движке, далее будут более развернутые выводы.

По результату обучения, модель делает хаотичные движения, дергается и кружится на месте, при этом длина эпизода маленькая, так как уровень проигрывается, по причине уничтожения агента пользователя.

2.3.2 Блок любопытства

Второй эксперимент проводился на модели с блоком любопытства, при этом в модели остается первый блок. В блоке нас интересуют два параметра: гамма – 0.99 и сила – 0.5.

Модель обучалась на 1 980 000 шагах, это заняло 5 часов. Модель достигает значения совокупного вознаграждения в 40 и средней продолжительности эпизода в 200 секунд. Агент в абсолютном большинстве игр побеждает, исследует игровую среду, активирует союзников и в целом делает все для успешного прохождения уровня.

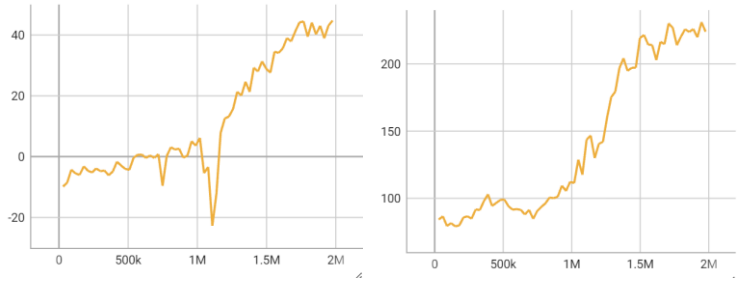


Рисунок 22 – Графики совокупного вознаграждения и продолжительности прохождения

2.3.4 Генеративное состязательное имитационное обучение

Третий эксперимент проводился на модели с блоком GAIL (Генеративное Состязательное Имитационное обучение), при этом в модели

остаются предыдущие блоки. В блоке нас интересуют два параметра: гамма – 0.9 и сила – 0.55. Для обучения был записан демо эпизод, состоящий из 10 прохождения уровня, сыгранных человеком.

Модель обучалась на 2 100 000 шагах, это заняло 4 часа. Модель достигает значения совокупного вознаграждения в 45 и средней продолжительности эпизода в 220 секунд. Агент не проигрывает, не трогает барьеры, уклоняется от снарядов и выполняет все необходимые действия для успешного прохождения уровня. При этом движения стали более осознанными, по сравнению с предыдущей моделью, практически нет случайных движений в сторону.

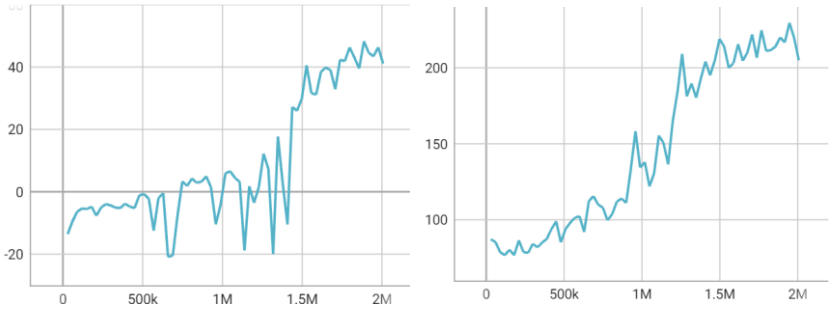


Рисунок 23 – Графики совокупного вознаграждения и продолжительности прохождения

2.3.5 Поведенческое клонирование

Четвертый эксперимент проводился на модели с блоком Behavioral Cloning, при этом в модели остаются предыдущие блоки. В блоке нас интересуют один параметр: сила – 0.5. Для обучения был записан демо эпизод, состоящий из 10 прохождения сыгранных человеком, который также использовался для обучения предыдущей модели.

Модель обучалась на 1 980 000 шагах, это заняло 5 часов. Модель достигает значения совокупного вознаграждения равной 64 и

продолжительности прохождения уровня в 214 секунд. Здесь по графикам можно увидеть, что агент достигает оценки предыдущих моделей уже к 500 000 шагу, результат достигается гораздо быстрее, конечная оценка в полтора раза больше предыдущих двух результатов. При этом агент получает меньше урона и активирует всех союзников, находящихся по пути.

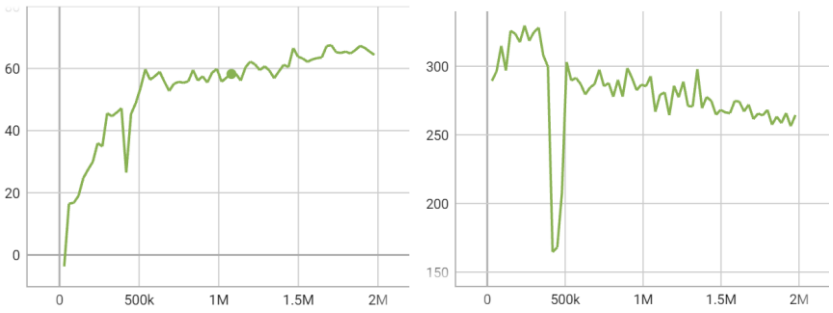


Рисунок 24 – Графики совокупного вознаграждения и продолжительности прохождения

3 Тестирование агента возможность на адаптации

3.1 Сравнение моделей

Выше были рассмотрены отдельно каждая из частей эксперимента. Теперь сравним результаты. На рисунках 25 и 26 представлены графики с результатами всех экспериментов.

По итогам экспериментов модель, использующая блок поведенческого клонирования показала наилучший результат, причем с отрывом. Стоит отметить, что важную роль в обучении такой модели, составляет человеческий фактор. От того, насколько качественные и разнообразные демоэпизоды будут записаны, на прямую зависит насколько быстро модель будет обучаться и насколько качественными будут результаты.

В эксперименте модели совершают приблизительно 2 000 000 шагов, поскольку при попытке дальнейшего обучения модель переобучается и показывает результат ниже, при этом значение функции потерь увеличивается.

Модель генеративного состязательного имитационного обучения и модель, использующая блок любопытства показали схожие результаты, как по совокупного вознаграждения, так и по средней продолжительности эпизода, что не соответствует документации. При обучении обеих моделей проверялись все рекомендуемые значения основных гиперпараметров с шагом в 0.1. Во всех успешных экспериментах модели показывали мало различимые результаты.

Как и ожидалось базовая модель не смогла показать результатов. Модель показывала результат близко к человеческому при прохождении уровней, в которых сильно упрощена игровая среда например: нет союзных юнитов и общее количество юнитов соперника не больше 3.

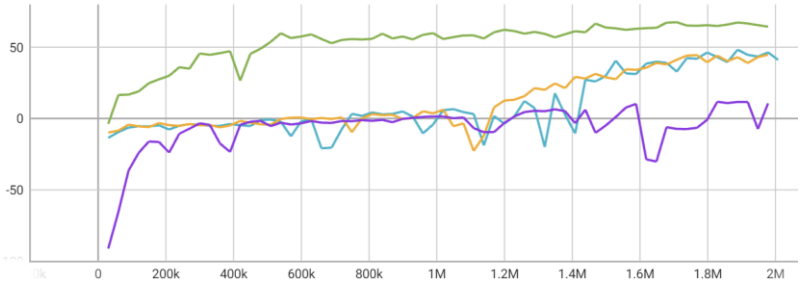


Рисунок 25 – График совокупного вознаграждения всех моделей

Добавлено примечание ([НБС4]): Добавить пустую строку

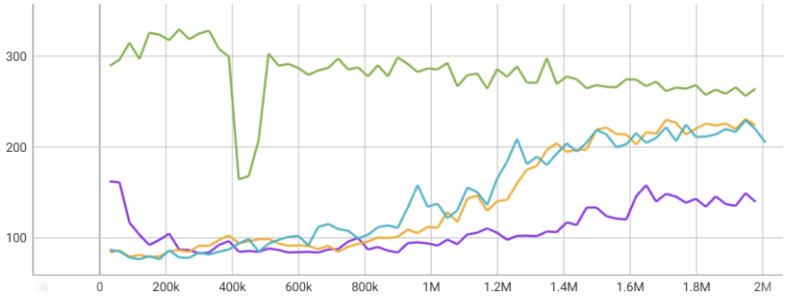


Рисунок 26 – Графики продолжительности прохождения всех моделей

Исходя из проведенного анализа, можно сделать вывод, что модель с блоком поведенческого клонирования оптимальна для решения поставленной задачи.

3.2 Тест модели

Для финального тестирования, как и в любой хорошей игре, агент “сразится” с боссом. В качестве эксперимента, возможности адаптации агента к новым условиям, была разработана система смены уровня. В конце каждого прохождения уровня в игровой среде меняются препятствия, их количество, размеры и угол поворота.

Система союзников была расширена классическим набором классов юнитов для игры в жанре top-downs shooter, и теперь состоит из трех вариаций

юнитов. Первый юнит взят из первого эксперимента. Второй юнит экипирован дробовиком, эффективен на короткой дистанции. Третий юнит экипирован гранатометом, эффективен при стрельбе по плотно находящимся врагам, так как при коллизии пули, наносит урон по области.

Количество классов соперников также увеличено до 3. Первый юнит так же взят из первого эксперимента. Второй противник имеет меньшее количество здоровья, увеличенную скорость, и экипирован ближним ударом вокруг себя. Третий юнит имеет большее количество здоровья, меньшую скорость и дробовик (эффективен на ближней дистанции).

Таким образом, агент оказывается в постоянно изменяющейся игровой среде, с постоянно меняющимся набором как союзников, так и врагов. Количество союзных юнитов равно 10, юнитов противника 20. Для эксперимента не увеличивается общее количество юнитов, чтобы не затягивать процесс обучения.

Для обучения модели в более сложным условиях, были изменены значения награды за действия, что привело к изменению среднего совокупного вознаграждения. Игрок в среднем проходит уровень на совокупное вознаграждение в диапазоне от 130 до 185. В зависимости от расположения союзников оценка может сильно меняться, так как не до всех из них агент или игрок успевает добраться и активировать.

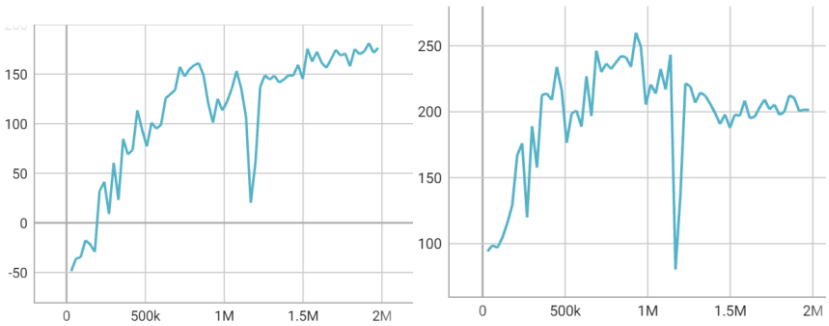


Рисунок 27 – Графики совокупной награды и продолжительности прохождения

На рисунке 27 представлены результаты обучения. В отличии от результатов предыдущих экспериментов, видно, что в этот раз модель обучалась и прогрессировала все 1 980 000 шагов. Совокупная награда модели составила 176.5, а время прохождения уровня составило 201 секунду. Учитывая тот факт, что количество соперников не изменилось, можно сделать вывод, что в более сложных условиях получилось обучить модель на более быстрое прохождение, по сравнению с другими экспериментами.

Агент в большинстве прохождений использует стратегии сбора всех соперников в кучу. В первую очередь подбирает максимальное количество близких союзников, экипированных базукой, для быстрой зачистки соперников в куче. Для выживания агент научился уворачиваться от пуль, и бежать к цели виляя в разные стороны, чтобы уменьшить шанс получения урона. При достаточном количестве активированных союзных юнитов, агент набирает дистанцию, при которой все могут использовать свою экипировку, а именно дистанцию достаточно большую для начала выстрелов, так как указана минимальная дистанция и не настолько большую дистанцию, чтобы снаряды долетали до врагов, так как у пуль есть дистанция полета, таким образом достигая максимальной эффективности отряда.

Важно отметить, что при наборе достаточной мощности агент не только набирает нужную дистанцию, но и останавливается на месте, если выстрелы успеют уничтожить соперников, а агент не успеет получить урон.

3.3 Внедрение модели при разработке проекта

Пакет Unity ML-Agents это инструмент для разработки моделей, который не предусматривает возможность использования одного готового решения для любого проекта. При внедрении данного подхода к реализации искусственного интеллекта соперника, лучшим решением будет изначально

подстроить архитектуру приложения под необходимые условия использования данного решения, которые обсуждались в главе 2.1.4.

План разработки не будет сильно отличаться от привычного плана разработки, изображенного на рисунке 28. Реализация классического подхода создания искусственного интеллекта соперника проходит на этапе разработки логики окружения и дополнительных механик.

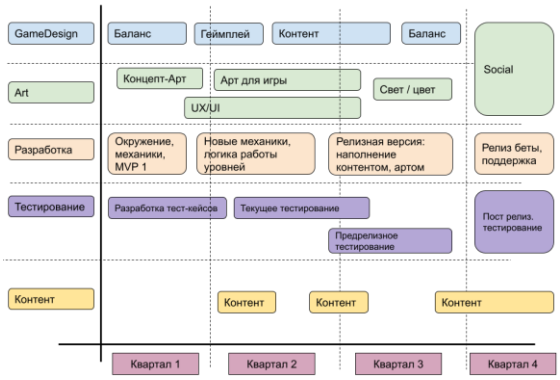


Рисунок 28 – План разработки игрового проекта

Время необходимое для разработки решения использующее методы машинного соизмеримо со временем разработки искусственного интеллекта соперника классическими методам. При этом проект выиграет время на этапе тестирования новых механик, баланса и в целом поддержки приложения. Стоит учесть, что при обучении агента нет необходимости в приобретении специальной техники, в целях увеличения производительной мощности, так как решение работает в среде игрового движка. Для процесса обучения хватит мощности стандартных устройств, используемых для разработки игровых проектов.

Также, возможна ситуация перехода от классического метода к использованию модели агента. Тема интеграции решения в уже готовый продукт так же поднималась в главе 2.1.4. При хорошей архитектуре

приложения нужны минимальные усилия для интеграции системы комнат, который нужны для ускорения процесса обучения модели агента.

С точки зрения дополнительных сложностей стоит отметить, что при формировании команды проекта, необходимо учитывать, что поиск разработчика высокого уровня, владеющего навыками по созданию искусственного интеллекта с помощью методов машинного обучения, будет проходить значительно сложнее и дольше, чем в классическом случае.

Таким образом, получается решение, внедрение которого возможно на разных этапах жизненного цикла приложения. При этом оно не требует дополнительных затрат и значительно упрощает тестирование новых механик и баланса, что может кратно сократить время реализации новых версий продукта.

ЗАКЛЮЧЕНИЕ

Добавлено примечание ([НБС5]): Заключение должно быть 2-3 страницы. Заголовки структурных элементов прописными буквами, См 43 методички

В рамках проведенной работы была разработана модель интеллектуального агента в игровой среде с использованием методов машинного обучения. Был разработан прототип игры в жанре top-down shooter, в котором есть возможность создавать разные классы юнитов как союзных, так и вражеских. Для разных классов юнитов была разработана система экипировки, с помощью которой были созданы несколько типов оружия. Реализован алгоритм обучения интеллектуального агента с использование методов машинного обучения, протестированы разные модели и выбрана оптимальные модель и техника обучения агента. Разработана усложненная игровая среда и новые классы юнитов как союзных, так и вражеских. Был обучен и протестирован интеллектуальный агент на предмет возможности адаптации к постоянно изменяющимся условиям в усложненной игровой среде.

Таким образом, была достигнута цель по разработке и проверке модели интеллектуального агента в игровой среде с использованием методов машинного обучения, с возможностью адаптации к новым условиям.

Проведенные эксперименты продемонстрировали, что выбранный подход позволяет добиться значительного улучшения показателей эффективности агента. В ходе работы было выявлено, что применение методов машинного обучения, таких как поведенческое клонирование и генеративное состязательное обучение, существенно повышает адаптивные способности агента в динамически изменяющейся игровой среде. Эти результаты подтверждают возможность использования разработанной модели для создания интеллектуальных агентов в других игровых жанрах и сценариях, требующих высокой степени адаптивности и автономности.

Получившаяся модель может быть использована не только в качестве инструмента для тестирования проектов на предмет ввода новых уровней, а именно их сложности, баланса и других интересующих метрик, но и, при

небольшой переработке основного геймплея, использоваться как искусственный интеллект соперника. При этом качество принимаемых решений и движения самого агента ничем не будет отличаться от действий обычного пользователя, что в свою очередь открывает большой спектр возможностей для создания ложного ощущения онлайн составляющей приложения. Это может значительно ускорить разработку прототипа приложения, доходя до этапа тестов на первичные метрики и тем самым сэкономить деньги и время студии разработки. Кроме того, использование разработанной модели может повысить уровень погружения пользователей в игровой процесс, благодаря чему улучшаются отзывы и общая удовлетворенность игроков. Повышенное качество искусственного интеллекта также способствует увеличению реиграбельности игры, что является важным фактором для долгосрочного успеха игрового проекта

В дальнейшем планируется протестировать возможность агента к обучению в игровых средах с усложненным дизайном уровней например: уровень с узкими проходами, уровень, состоящий из коридоров и т.д. Также, в планах добавить механики вражеских отрядов и протестировать модель на решение задач мультиагентого обучения. Провести визуальные улучшения. Проверить совместимость с другими платформами: WebGL, Android, а также производительность решения по сравнению с классическими реализациями искусственного интеллекта в игровых проектах реализованных, с помощью игрового движка Unity. Дополнительные исследования будут направлены на оптимизацию алгоритмов обучения для уменьшения времени обучения и повышения эффективности использования вычислительных ресурсов. Планируется внедрить механизмы непрерывного обучения, позволяющие агенту адаптироваться к новым условиям без необходимости полной переобучения. Это позволит сократить время разработки и улучшить динамическую адаптацию агента к новым игровым сценариям.

Работа по созданию интеллектуальных агентов, адаптирующихся к изменяющимся условиям, представляет значительный шаг вперед в области

разработки искусственного интеллекта для игр. Этот подход открывает новые возможности для создания более реалистичных и увлекательных игровых опытов, что является ключевым фактором в конкурентной игровой индустрии.

В итоге создана и продемонстрирована модель интеллектуального агента на базе Unity ML-Agent, с использованием методов машинного обучения, с возможностью адаптации к новым условиям, отвечающая всем требованиям и обладающая достаточной гибкостью для внесения изменений и улучшений, для дальнейшей работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Engelbrecht, D. Introduction to Unity ML-Agents: Understand the Interplay of Neural Networks and Simulation Space Using the Unity ML-Agents Package. Introduction to Unity ML-Agents / D. Engelbrecht Google-Books-ID: f4RwzwEACAAJ Apress, 2023. – 204 p.– ISBN 978-1-4842-8997-6. – Text : direct.

2. Haarnoja, T. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor // Proceedings of the 35th International Conference on Machine Learning International Conference on Machine Learning. PMLR, 2018. Soft Actor-Critic. – P. 1861-1870.

3. Lanham, M. Learn Unity ML-Agents – Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games. Learn Unity ML-Agents – Fundamentals of Unity Machine Learning / M. Lanham Google-Books-ID: OMNiDwAAQBAJ Packt Publishing Ltd, 2018. – 197 p.– ISBN 978-1-78913-186-4. – Text : direct.

4. Nandy, A. Unity ML-Agents / A. Nandy, M. Biswas // Neural Networks in Unity: C# Programming for Windows 10 / eds. A. Nandy, M. Biswas. – Berkeley, CA: Apress, 2018. – P. 27-67.

5. Что такое обучение с подкреплением? — Объяснение обучения с подкреплением — AWS – Узнайте, что такое обучение с подкреплением, как и почему компании используют обучение с подкреплением и как использовать обучение с подкреплением в AWS. – URL: https://aws.amazon.com/ru/what-is/reinforcement-learning/ (дата обращения: 20.03.2024) – Текст: электронный.

6. А.н, Ш. АЛГОРИТМЫ РАННЕГО ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ // Экономика и социум. 2023. № 6-2 (109). – С. 1124-1126.

7. Юрьевич, Ч. Д. Машинное обучение / Ч.Д. Юрьевич, И.В. Вадимович – Текст : непосредственный. // Наука, техника и образование. 2018. № 5 (46). – С. 85-87.

8. Lee, H. Character Behavior Automation Using Deep Reinforcement Learning / H. Lee, M.K. Dahouda, I. Joe – Текст : непосредственный. // IEEE Access. 2023. Т. 11. – С. 101435-101442.

9. Schulman, J. Proximal Policy Optimization Algorithms // ArXiv. 2017.

10. Youssef, A. E. Building your kingdom Imitation Learning for a Custom Gameplay Using Unity ML-agents // 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). 2019. – С. 0509-0514.

11. Deep Reinforcement Learning in Agents' Training: Unity ML-Agents | SpringerLink. – URL: https://link.springer.com/chapter/10.1007/978-3-031-06527-9_39 (дата обращения: 18.12.2023) – Текст: электронный.

12. Fruit Picking Robot Arm Training Solution Based on Reinforcement Learning in Digital Twin | River Publishers Journals & Magazine | IEEE Xplore. – URL: https://ieeexplore.ieee.org/document/10255409 (дата обращения: 20.03.2024) – Текст: электронный.

13. Inversion of Control in Game Development : Strange IoC - Theseus. – URL: https://www.theseus.fi/handle/10024/264878 (дата обращения: 14.05.2024) – Текст: электронный.

14. Proximal Policy Optimization — Spinning Up documentation. – URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html (дата обращения: 20.03.2024) – Текст: электронный.

15. Proximal Policy Optimization (PPO) – We're on a journey to advance and democratize artificial intelligence through open source and open science. – URL: https://huggingface.co/blog/deep-rl-ppo (дата обращения: 20.03.2024) – Текст: электронный.

16. Unity ML-Agents Toolkit. – URL: https://unity-technologies.github.io/ml-agents/ (дата обращения: 20.03.2024) – Текст: электронный.

17. Unity-Technologies/ml-agents. – Unity Technologies, 2024.

18. Overview of ASP.NET Core MVC ardalis – Learn how ASP.NET Core MVC is a rich framework for building web apps and APIs using the Model-View-Controller design pattern. – URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0 (дата обращения: 22.03.2024) – Текст: электронный.

19. Parviainen, N. Dependency Injection in Unity3D / N. Parviainen – Text : direct.

20. modesttree/Zenject. – Modest Tree Media Inc, 2024.

21. Марк, С. Внедрение зависимостей на платформе .NET. 2-е издание / С. Марк, ван Д. Стивен Google-Books-ID: uKceEAAAQBAJ «Издательский дом „«Питер»“», 2020. – 608 с.– ISBN 978-5-4461-1166-4. – Текст : непосредственный.

22. Brathwaite, B. Challenges for game designers: non-digital exercises for video game designers. Challenges for game designers / B. Brathwaite, I. Schreiber ; Nachdr. – Boston, Mass.: Course Technology, Cengage Learning, 2011. – 317 p.– ISBN 978-1-58450-580-8. – Text : direct.

23. Build More Engaging Games with ML Agents | Unity U. Technologies – Unity Machine Learning Agents is an open-source virtual game character creation software that requires no prior experience. Download for free today! – URL: https://unity.com/products/machine-learning-agents (дата обращения: 12.05.2024) – Текст: электронный.

24. Robotics Simulation | Unity U. Technologies – Unity’s robotics simulation allows you to verify your program before implementing them in a robot. Download our robotics demos to learn more. – URL: https://unity.com/solutions/automotive-transportation-manufacturing/robotics (дата обращения: 12.05.2024) – Текст: электронный.

25. Emergent tool use from multi-agent interaction – We’ve observed agents discovering progressively more complex tool use while playing a simple game of hide-and-seek. Through training in our new simulated hide-and-seek environment, agents build a series of six distinct strategies and counterstrategies, some of which

we did not know our environment supported. The self-supervised emergent complexity in this simple environment further suggests that multi-agent co-adaptation may one day produce extremely complex and intelligent behavior. – URL: https://openai.com/index/emergent-tool-use/ (дата обращения: 12.05.2024) – Текст: электронный.

26. Игоревич, Д. К. Возможное применение нейронных сетей в играх в будущем / Д.К. Игоревич – Текст : непосредственный. // Наука, образование и культура. 2018. № 10 (34). – С. 12-13.

27. Коробов, Д. А. СОВРЕМЕННЫЕ ПОДХОДЫ К ОБУЧЕНИЮ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ В СРЕДЕ Atari / Д.А. Коробов, С.А. Беляев – Текст : непосредственный. // Программные продукты и системы. 2018. Т. 31. № 2. – С. 284-290.

28. Олегович, А. А. МОДЕЛИРОВАНИЕ ПОВЕДЕНИЯ АГЕНТОВ ДЛЯ РЕАЛИЗАЦИИ ИГРОВОГО ИСКУССТВЕННОГО ИНТЕЛЛЕКТА // Прикаспийский журнал: управление и высокие технологии. 2020. № 2 (50). – С. 85-99.

29. О.м, Р. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ВВЕДЕНИЕ / Р. О.м, Ш. А.д – Текст : непосредственный. // Теория и практика современной науки. 2020. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ. № 1 (55). – С. 477-482.

30. Andersen, P.-A. Towards a Deep Reinforcement Learning Approach for Tower Line Wars. Т. 10630 / P.-A. Andersen, M. Goodwin, O.-C. Granmo arXiv:1712.06180 [cs] 2017.

31. АНАЛИЗ ЧИСТОЙ АРХИТЕКТУРЫ GOLANG REST API С ВНЕДРЕНИЕМ ЗАВИСИМОСТЕЙ, СЛЕДУЯ ПРИНЦИПАМ SOLID. – URL: https://cyberleninka.ru/article/n/analiz-chistoy-arhitektury-golang-rest-api-s-vnedreniem-zavisimostey-sleduya-printsipam-solid/viewer (дата обращения: 12.05.2024) – Текст: электронный.

32. Unity-Technologies/Unity-Robotics-Hub. – Unity Technologies, 2024.

33. Лаура, Г. Глубокое обучение с подкреплением: теория и практика на языке Python. Глубокое обучение с подкреплением / Г. Лаура, К. В. Лун

Google-Books-ID: CThcEAAAQBAJ «Издательский дом „«Питер»“», 2021. – 416 с.– ISBN 978-5-4461-1699-7. – Текст : непосредственный.

34. Goodfellow, I. Deep learning / I. Goodfellow Cambridge, Massachusetts : The MIT Press, 2016. – 810 с.– ISBN 978-0-262-03561-3. – Текст : непосредственный.

35. Reinforcement Learning An Introduction, Richard S. Sutton And Andrew G. Barto.

36. Межидов, А. Р. Внедрение зависимостей. Ninject / А.Р. Межидов – Текст : непосредственный.

37. Сергеевна, Б. А. Метод повышения эффективности эволюционных алгоритмов с помощью обучения с подкреплением / Б.А. Сергеевна, Б.М. Викторович – Текст : непосредственный. // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 5 (81). – С. 115-119.

38. Arulkumaran, K. Deep Reinforcement Learning: A Brief Survey // IEEE Signal Processing Magazine. 2017. Т. 34. Deep Reinforcement Learning. № 6. – С. 26-38.

39. Greg Brockman. OpenAI Gym / Greg Brockman, Vicki Cheung, Ludwig Pettersson [и др.] arXiv.org, 2016.

40. Kober, J. Reinforcement Learning in Robotics: A Survey / J. Kober, J. Bagnell, J. Peters – Текст : непосредственный. // The International Journal of Robotics Research. 2013. Т. 32. Reinforcement Learning in Robotics. – С. 1238-1274.

41. Lillicrap, T. P. Continuous control with deep reinforcement learning. 2019.

42. Sabharwal, A. S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Third Edition. / A. Sabharwal, B. Selman – Текст : непосредственный. // Artif. Intell. 2011. Т. 175. S. Russell, P. Norvig, Artificial Intelligence. – С. 935-937.

43. Sabharwal, A. S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Third Edition. / A. Sabharwal, B. Selman – Текст : непосредственный. // Artif. Intell. 2011. Т. 175. S. Russell, P. Norvig, Artificial Intelligence. – С. 935-937.

44. Silver, D. Mastering the game of Go with deep neural networks and tree search // Nature. 2016. Т. 529. – С. 484-489.

45. Deep Learning. – URL: https://www.deeplearningbook.org/ (дата обращения: 23.05.2024) – Текст: электронный.