СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc167202604)

[1. АНАЛИЗ СУЩЕСТВУЮЩИХ СИСТЕМ, ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ 6](#_Toc167202605)

[1.1 АНАЛИЗ СУЩЕСТВУЮЩИХ СИСТЕМ 6](#_Toc167202606)

[1.2 АНАЛИЗ АЛГОРИТМОВ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ 7](#_Toc167202607)

[1.3 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 11](#_Toc167202608)

[1.4 ПОСТАНОВКА ЗАДАЧИ 12](#_Toc167202609)

[2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ 14](#_Toc167202610)

[2.1СРЕДСТВА РЕАЛИЗАЦИИ 14](#_Toc167202611)

[2.2ОБЩЕЕ ОПИСАНИЕ СТРУКТУРЫ ПРОГРАМ 17](#_Toc167202612)

[2.3ПРОЕКТИРОВАНИЕ ПРОГРАММЫ 22](#_Toc167202613)

[3.РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ 26](#_Toc167202614)

[3.1 РЕАЛИЗАЦИЯ ПРОГРАММНЫХ СРЕДСТВ 26](#_Toc167202615)

[3.2 ОБУЧЕНИЕ И ТЕСТИРОВАНИЕ МОДЕЛЕЙ 33](#_Toc167202616)

[4.ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ 37](#_Toc167202617)

[4.1 ИСХОДНЫЕ ДАННЫЕ ДЛЯ РАСЧЕТА ЭКОНОМИЧЕСКОГО ЭФФЕКТА 37](#_Toc167202618)

[4.2 РАСЧЕТ ОБЪЕМА ФУНКЦИЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 38](#_Toc167202619)

[4.2 РАСЧЕТ ПОЛНОЙ СЕБЕСТОИМОСТИ ПРОГРАММНОГО ПРОДУКТА 39](#_Toc167202620)

[4.4 РАСЧЕТ ЦЕНЫ И ПРИБЫЛИ ПО ПРОГРАММНОМУ ПРОДУКТУ 45](#_Toc167202621)

[5. ЭНЕРГОСБЕРЕЖЕНИЕ 47](#_Toc167202622)

[ЗАКЛЮЧЕНИЕ 51](#_Toc167202623)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 52](#_Toc167202624)

[СПИСОК СОКРАЩЕНИЙ 54](#_Toc167202625)

**ВВЕДЕНИЕ**

В современном мире нейронные сети уже не новинка. Они уже используются во многих сферах нашей жизни, их встраивают в системы управления автомобилем, при помощи их прогнозируется погода, в некоторых странах нейронные сети заменяют работников на предприятиях. В настоящее время нейронные сети вошли в практику везде, где это необходимо для решения задач прогнозирования, распознавания образов, классификации или контроля. Эти мощные инструменты машинного обучения позволяют компаниям и организациям решать сложные задачи, которые ранее казались невозможными. Применение нейронных сетей в работе приводит к увеличению производительности, снижению затрат и разработке инновационных продуктов.

Одним из наиболее ярких примеров применения нейронных сетей является обработка текста и анализ естественного языка. Нейросети способны анализировать текстовые данные, извлекать смысл и контекст, классифицировать тексты и даже генерировать текст в стиле человеческого написания. Это пригодно для автоматической обработки текстовых данных, как например, в области клиентского обслуживания, где нейронные сети могут использоваться для анализа отзывов клиентов и определения их настроения, а также для автоматической генерации текстовых ответов.

В данной работе будет рассмотрена разработка программной платформы по созданию цифровых ассистентов (чат-ботов), которые будут основываться на таких генерирующих нейронных сетях.

Для достижения этой цели будет использована библиотека LangChain для языка программирования python, которая облегчит взаимодействие с нейронными сетями и позволит подгружать контекст диалога для улучшения способностей искусственного интеллекта, в частности его знаний конкретной темы разговора.

Таким образом, данная работа будет представлять собой практическое исследование в области применения нейронных сетей и создания легко расширяемых программных продуктов.

В данной работе главная цель – создание платформы, которая позволит разветвлять области её применения без больших затрат сил по переделыванию методов взаимодействия с нейронными сетями и их пользовательским интерфейсом. Каждая ветка этой платформы будет уметь отвечать на разные темы, которые будут интересовать пользователя. Ответы будут подкреплены учебниками или другими документами в электронном формате.

# АНАЛИЗ СУЩЕСТВУЮЩИХ СИСТЕМ, ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ

В этой главе будут введены основные понятия предметной области, проанализируем существующие системы, выделим их плюсы и минусы, рассмотрим различные библиотеки, которые позволят выполнить цель работы.

* 1. **АНАЛИЗ СУЩЕСТВУЮЩИХ СИСТЕМ**

На данный момент существует узкий круг систем, которые выполняют функции схожие с разрабатываемой платформой. На начало разработки их практически не существовало, однако подходя к концу цикла разработки стали появляться схожие системы от больших компаний-разработчиков. Рассмотрим несколько известных аналогов и их характеристики:

* **Coze:**
  + *Плюсы:*
    - Полностью бесплатный хост бота без ограничений по сообщениям.
    - Минималистичный пользовательский интерфейс, нету необходимости в долгом изучении программы перед использованием.
    - Позволяет загружать таблицы в базу знаний бота.
  + *Минусы:*
    - Чат с ботом доступен только на определённых платформах, нельзя разместить его в пользовательском API.
* **Popai:**
  + *Плюсы:*
    - Пользовательский интерфейс красив и лёгок в использовании.
    - Присутствует возможность бесплатного использования.
  + *Минусы:*
    - Использование бесплатно сильно ограничено, платная подписка достаточно дорогая, стоимость которой составляет 33 доллара США.
* **Flowxo:**
  + *Плюсы:*
    - Сервис поддерживает большой спектр моделей нейронных сетей.
    - Имеет возможность бесплатного использования.
  + *Минусы:*
    - Бесплатно пользоваться можно лишь периодом в 14 дней, после чего аккаунт перестаёт работать.
    - Отсутствует возможность вывести общение с ботом через API.

Хотя указанные платформы демонстрируют впечатляющий функционал, применить их для целей данной работы невозможно из-за отсутствия решения, которое позволит разместить бота в виде API для встраивания в пользовательский интерфейс за разумную цену. Для достижения целей работы необходимо создать собственное решение, которое будет отвечать всем необходимым требованиям.

## 1.2 АНАЛИЗ АЛГОРИТМОВ ПОДАЧИ КОНТЕКСТА

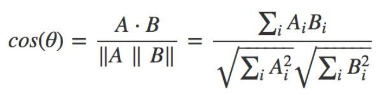
Подача (подгрузка) контекста – важная часть работы цифрового ассистента. Процесс подачи контекста заключается в поиске текстовой информации, которая поможет ассистенту ответить на поставленный человеком вопрос при помощи внешних источников, например научных статей, найденных в интернете, либо электронных документов, загруженных в базу знаний ассистента заранее. Данный процесс принято называть RAG (Retrieval Augmented Generation), однако далее будет написано «подача контекста».

Рассмотрим шаги, которые необходимо выполнить для реализации механизма подачи контекста в цифровых ассистентах:

* **Сборка основы базы знаний:** Собираются необходимые документы, которые будут присутствовать в базе знаний для подкрепления ответов бота. В основном это различные документы формата «.pdf», которые помещаются в отдельную папку компьютера, на котором планируется запуск цифрового ассистента.
* **Разделение текста на части:** Весь тест из каждого документа считывается и обрезается в куски размером от 1300 до 1600 слов, в зависимости от длины последнего предложения.
* **Векторизация текста:** Каждый кусок текста проходит через специальную функцию векторизации. Эта функция векторизации преобразует текст в набор математических векторов, которые отражают семантический смысл предложения (его смысл).
* **Хранение векторов:** Все преобразованные пары текста и вектора хранятся в векторной базе данных, что и является базой знаний цифрового ассистента.
* **Получение запроса от пользователя:** Как только система получает вопрос от пользователя, его вопрос преобразуется по точно такой же функции векторизации и полученные векторы сравниваются с теми, что хранятся в базе знаний. Обычно для этого используется косинусная схожесть между векторами. Самый близкий вектор в базе знаний по сравнению с вопросом и является необходимым нам куском текста.
* **Подача контекста:** Этот найденный в базе знаний кусок текста помещается в контекст для нейронной сети вместе с вопросом пользователя, после чего полученный ответ после генерации ассистентом можно предоставить пользователю. Таким образом, цифровой ассистент видит не только вопрос пользователя, но и предоставленную для ответа на этот вопрос информацию, произошла подача контекста.

Для успешного выполнения описанных ранее шагов, необходимо определиться с используемыми алгоритмами и функциями, в данном случае конкретно с алгоритмом определения схожести векторов и функцией векторизации.

В области нейронных сетей схожесть двух векторов чаще всего определяют путём нахождения косинусной схожести (cosine similarity), так как результатом этого является число в диапазоне от 0 до 1 в каждом случае. Косинусная схожесть определяется следующей формулой:



где:

А – первый вектор.

В – второй вектор.

Аi – очередной компонент вектора А.

Bi – очередной компонент вектора В.

Кроме алгоритма определения схожести векторов, необходимо выбрать функцию векторизации. В сети интернет размещено в открытом доступе огромное количество таких функций. Они представляют из себя преобразующие нейронные сети, которые позволяют захватить смысловую нагрузку последовательных данных, будь это текст или картинка. В случае данной работы необходимо выбрать один из таких преобразователей, который воспринимает на вход текст и выдаёт n-размерный вектор.

Исходя из результатов бенчмарков пользователей интернет-ресурса HuggingFace (известный веб-сайт посвящённый открытым нейронным сетям), было решено в работе использовать функцию под названием «sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2». Данная функция векторизации преобразует параграфы и предложения в вектор размером 384 и может быть использована для семантического поиска. Выбор обусловлен хорошими результатами бенчмарков (3 место в таблице лидеров), большим количеством скачиваний (отражает популярность модели), а также наличием поддержки русского, белорусского, английского и немецкого языков.

* 1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

Предметная область данной работы охватывает применение искусственного интеллекта (ИИ) в контексте файтинг игр, где игроки управляют персонажами, сражаясь друг с другом в виртуальных баталиях. Основные понятия и термины, используемые в этой области, включают в себя:

* **Искусственный интеллект (ИИ):** это система, которая способна воспринимать свою среду и принимать меры, чтобы максимизировать шансы на успешное достижение своих целей, а также интерпретировать и анализировать данные таким образом, чтобы она обучалась и адаптировались по мере развития.
* **Агент**: Агент представляет собой программу искусственного интеллекта, которая учится играть в SF2SCE, оптимизируя свою стратегию на основе полученного опыта и награды. В данной программе агент обучается с использованием алгоритма PPO.
* **Обучение с подкреплением:** Метод машинного обучения, при котором агент (в данном контексте – игровой персонаж) обучается путем взаимодействия с окружающей средой, принимая действия и получая обратную связь в виде награды или штрафа за свои действия.
* **Игровая среда:** В SF2SCE среда представляет собой виртуальный мир игры, в котором действует агент. В этой программе среда моделируется с помощью библиотеки Gym Retro, которая обеспечивает доступ к ретро-играм и позволяет создавать игровые среды для обучения агентов.
* **Предобработка данных:** Процесс подготовки данных для обучения модели ИИ. Включает в себя преобразование изображений из игровой среды в оттенки серого и изменение их размера для упрощения обучения модели.
* **Наблюдение**: Наблюдение представляет собой состояние игровой среды, доступное агенту в каждый момент времени. В SF2SCE наблюдение включает в себя текущий кадр игры, который агент использует для принятия решений.
* **Действие**: Действие представляет собой действие, которое агент выбирает в ответ на текущее наблюдение. В SF2SCE действия могут включать в себя различные комбинации ударов, блокировку, прыжки и специальные атаки.
* **Награда**: Награда представляет собой численную оценку, которую агент получает за свои действия. В SF2SCE награда может быть определена, например, как количество очков, заработанных агентом в процессе игры.
* **Алгоритмы обучения:** Методы и подходы, используемые для обучения модели ИИ. В данной работе применяются алгоритмы PPO для обучения модели, способной играть в файтинг игры.
* **Оптимизация гиперпараметров:** Процесс настройки параметров модели для достижения наилучшей производительности. В данной работе используется библиотека Optuna для оптимизации гиперпараметров алгоритма PPO.
* **Обучение**: Процесс, в ходе которого агент адаптирует свою стратегию на основе полученного опыта. В данной программе агент обучается с использованием алгоритма PPO, который позволяет ему итеративно улучшать свою стратегию, максимизируя суммарную награду.

Эти основные концепции играют ключевую роль в контексте работы для разработки алгоритма ИИ, способного эффективно управлять персонажем в файтинг играх.

* 1. **ПОСТАНОВКА ЗАДАЧИ**

В данной работе требуется разработать программу для обучения агента в игре SF2SCE на симуляции консоли Genesis. Программа должна обеспечивать следующий функционал:

* Загрузку и инициализацию игровой среды SF2SCE.
* Предобработку изображений игрового окна, включающую конвертацию в черно-белый формат, изменение разрешения и выделение различий между кадрами.
* Обучение агента с использованием алгоритма обучения с подкреплением PPO на основе данных о состояниях игры, выполненных действиях и полученных наградах.
* Тестирование обученного агента в игровой среде для оценки его производительности и эффективности в сражениях.
* Оптимизацию гиперпараметров алгоритма PPO с использованием библиотеки Optuna для повышения производительности и качества обучения агента.
* Сохранение промежуточных результатов обучения для последующего анализа и возможности продолжения обучения с сохраненного момента.

1. **ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ**

В этой главе спроектируем программную систему. Для этого опишем ее общую структуру и в подробностях рассмотрим все подсистемы и выберем средства программирования для реализации программы.

* 1. **СРЕДСТВА РЕАЛИЗАЦИИ**

В процессе разработки дипломной работы был выбран язык программирования Python. Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. В Python имеются встроенные типы: булевый, строка, Unicode-строка, целое число произвольной точности, число с плавающей запятой, комплексное число и некоторые другие. Из коллекций в Python встроены: список, кортеж (неизменяемый список), словарь, множество и другие. Все значения являются объектами, в том числе функции, методы, модули, классы.

В качестве интегрированной среды разработки (IDE) был использован PyCharm. PyCharm — интегрированная среда разработки для языка программирования Python. PyCharm был выпущен на рынок интегрированных сред разработки для Python для создания конкуренции с PyDev и более распространённой среды разработки Komodo IDE. Бета-версия была выпущена в июле 2010 года, версия 1.0 была выпущена тремя месяцами позже. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией JetBrains на основе IntelliJ IDEA.

PyCharm поддерживает:

* Статический анализ кода, подсветка синтаксиса и ошибок.
* Навигация по проекту и исходному коду: отображение файловой структуры проекта, быстрый переход между файлами, классами, методами и использованиями методов.
* Рефакторинг: переименование, извлечение метода, введение переменной, введение константы, подъём и спуск метода и т. д.
* Инструменты для веб-разработки с использованием фреймворка Django.
* Встроенный отладчик для Python.
* Встроенные инструменты для юнит-тестирования.
* Разработка с использованием Google App Engine.
* Поддержка систем контроля версий: общий пользовательский интерфейс для Mercurial, Git, Subversion, Perforce и CVS с поддержкой списков изменений и слияния.

Для проведения экспериментов и исследований, а также для демонстрации результатов работы алгоритма использовались Jupyter Notebook. Jupyter Notebook был выбран как основная среда разработки, что обусловлено рядом ключевых преимуществ данного инструмента. Jupyter Notebook предоставляет возможность интерактивного выполнения кода, что особенно ценно при тестировании отдельных фрагментов алгоритмов и наблюдении за их выполнением в реальном времени. Эта особенность существенно упрощает процесс разработки и отладки.

Одним из значимых преимуществ Jupyter Notebook является поддержка визуализации данных через интеграцию с популярными библиотеками, такими как matplotlib и seaborn. Это позволяет наглядно представлять результаты анализа и работы алгоритмов прямо в рабочем документе, что делает процесс исследования более интуитивным и понятным.

Важной характеристикой Jupyter Notebook является возможность документирования исследовательского процесса. Поддержка разметки Markdown и LaTeX позволяет оформлять записные книжки с подробными пояснениями, выводами и математическими формулами, делая их содержание доступным для анализа и представления результатов.

Среда также способствует совместной работе над проектами благодаря удобству распространения записных книжек среди коллег или в публичном доступе. Это облегчает обмен знаниями и коллаборативную разработку.

Jupyter Notebook поддерживает работу с множеством языков программирования, что расширяет возможности для мультиязыкового программирования и исследований. Интеграция с различными инструментами и платформами, включая Google Colab, предоставляет доступ к ресурсам облачных вычислений и упрощает процесс разработки.

Наконец, расширяемость Jupyter Notebook через множество доступных расширений добавляет дополнительные функциональные возможности, такие как версионирование и автоматическое форматирование кода, что делает рабочий процесс более организованным и эффективным.

В качестве основной библиотеки для создания и управления средой обучения была использована библиотека gym от OpenAI. Она предоставляет унифицированный интерфейс для различных симулированных сред, что позволяет стандартизировать процесс разработки и тестирования алгоритмов обучения с подкреплением.

Для реализации алгоритма PPO была выбрана библиотека stable\_baselines3, предлагающая готовые реализации различных алгоритмов обучения с подкреплением, включая PPO. Эта библиотека обеспечивает эффективную работу с нейронными сетями и автоматизацию процессов обучения.

Для создания и обучения нейронных сетей использовалась библиотека PyTorch, которая позволяет гибко управлять процессами автоматического дифференцирования и оптимизации, что критически важно для эффективного обучения глубоких нейронных сетей.

Для визуализации процесса обучения и анализа результатов использовались инструменты TensorBoard и matplotlib, которые позволяют отслеживать метрики обучения в реальном времени и строить графики и диаграммы для более глубокого анализа производительности модели.

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для C++, Haskell, Java, Go и Swift.

Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0.

TensorFlow 9 ноября 2015 года был открыт для свободного доступа. TensorFlow является системой машинного обучения Google Brain второго поколения. В то время как эталонная реализация работает на единичных устройствах, TensorFlow может работать на многих параллельных процессорах, как CPU, так и GPU, опираясь на архитектуру CUDA для поддержки вычислений общего назначения на графических процессорах. TensorFlow доступна для 64-разрядных Linux, macOS, Windows, и для мобильных вычислительных платформ, включая Android и iOS.

Вычисления TensorFlow выражаются в виде потоков данных через граф состояний. Название TensorFlow происходит от операций с многомерными массивами данных, которые также называются «тензорами».

Matplotlib — это библиотека языка программирования Python для создания графиков и визуализации данных. Она предоставляет широкий спектр инструментов для создания различных типов графиков и диаграмм, включая линейные и точечные диаграммы, гистограммы, диаграммы разброса, контурные и поверхностные графики, а также карты и анимации. Matplotlib может использоваться для визуализации различных типов данных, включая временные ряды, пространственные данные, данные социальных сетей, данные научных исследований и многое другое. Она также предоставляет возможности для настройки внешнего вида графиков, включая цветовые схемы, шрифты, легенды и т.д. Она может быть использована для визуализации сигналов ЭЭГ, представления результатов анализа эмоций и оценки моделей глубокого обучения.

Таким образом, выбор средств программирования был обусловлен необходимостью использования современных, надёжных и проверенных инструментов, которые обеспечивают гибкость и масштабируемость при разработке сложных систем машинного обучения.

* 1. **ОБЩЕЕ ОПИСАНИЕ СТРУКТУРЫ ПРОГРАМ**

Необходимо спроектировать структуру программы, которая удовлетворяла бы поставленным задачам. Для этого последовательно выдели все основные подсистемы, которые способны решить эти задачи.

Подсистема «Игровая среда». Цель данной работы состоит в создании и обучении агента искусственного интеллекта для файтинг игр на примере SF2SCE. Поэтому необходимо иметь подсистему, отвечающую за взаимодействие со средой, симулирующей SF2SCE.

Подсистема «Оптимизация гиперпараметров». Так как в этой работе происходит обучение агента искусственного интеллекта на основе обучения с подкреплением, то важной частью является оптимизация гиперпараметров для улучшения производительности агента, соответственно нужно иметь подсистему, отвечающую за эти процессы.

Подсистема «Тренировка и оценка». Из-за специфики обучения, а именно большого количества обучаемых моделей, различных по количеству эпох обучения и гиперпараметрам, нужно иметь подсистему, отвечающую за оценку моделей и выбор лучшей из них.

Подсистема «Тестирование модели». Завершающий этап – тестирование обученной модели путем ее взаимодействия с игровой средой и отображения результатов.

Уточненная схема взаимодействия подсистем представлена на рисунке 2.1.1.

Рисунок 2.1.1 – Схема взаимодействия подсистем «Игровая среда», «Оптимизация гиперпараметров», «Тренировка и оценка» и «Тестирование модели»

Далее выделим подсистемы, взаимодействующие с «Игровой средой».

Подсистема «Создание среды». При инициализации создается объект игровой. Эта среда используется для взаимодействия с игрой.

Подсистема «Определение пространств действий и наблюдений». Также определяются пространства действий и наблюдений в виртуальной среде.

Подсистема «Взаимодействие с игровой средой». Есть несколько методов для взаимодействия с игровой средой: сброс среды в начальное состояние, принятие действий, отображение текущего состояния игры, закрытие игровой среду.

Подсистема «Предобработка». Предварительно, также нужно предварительно обработать изображение игры, используемое для обучения, путем уменьшенья разрешения и изменение цветовых каналов RGB на один канал, черно-белый.

Уточненная схема взаимодействия подсистем представлена на рисунке 2.1.2.

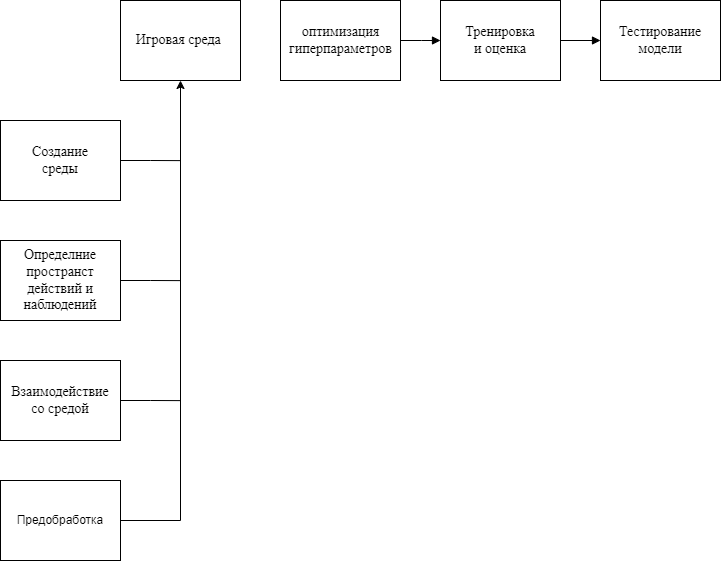


Рисунок 2.1.2 – Схема взаимодействия подсистем, уточненная в соответствии с подсистемами, реализующими взаимодействие с игровой средой

Далее рассмотрим подсистемы «Оптимизация гиперпараметров».

Подсистема «Функция оптимизации гиперпараметров»: включает определение пространства поиска и целевой метрики для оптимизации.

Подсистема «Оценщик производительности»: включает оценку производительности моделей.

Подсистема «Журналирование и сохранение результатов»: включает запись результатов каждого эксперимента, включая значения гиперпараметров и их производительность, для дальнейшего анализа.

Уточненная схема взаимодействия подсистем представлена на рисунке 2.1.3.

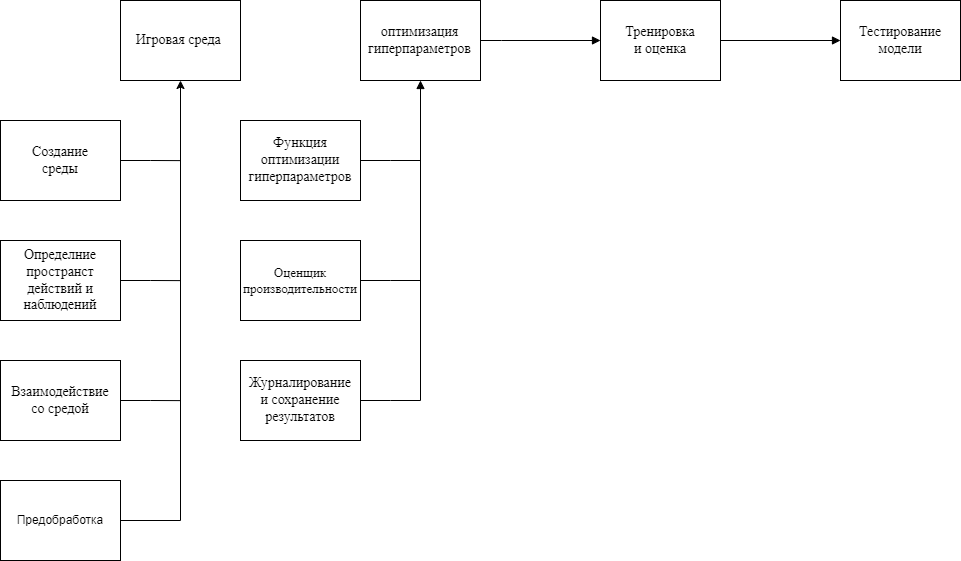


Рисунок 2.1.3 – Схема взаимодействия подсистем, уточненная в соответствии с подсистемами, реализующими оптимизацию гиперпараметров

Подсистемы «Тренировка и оценка».

Подсистема «Тренировка модели». Настройка алгоритма обучения: Создание модели с помощью алгоритма PPO и указание гиперпараметров для обучения. Обучение модели: использует созданную модель и окружение для обучения на определенное количество временных шагов.

Подсистема «Оценка модели». Оценка производительности: включает оценку производительности обученной модели на основе заданных метрик, таких как средняя награда. Тестирование модели: позволяет протестировать обученную модель в игровой среде и оценить ее поведение.

Подсистема «Журнал результатов». Ведение журнала: записывает результаты обучения и оценки производительности модели для анализа и последующего использования. Сохранение лучших моделей: сохраняет параметры и веса лучших моделей, чтобы их можно было повторно использовать или дальше обучать.

Уточненная схема взаимодействия подсистем представлена на рисунке 2.1.4.

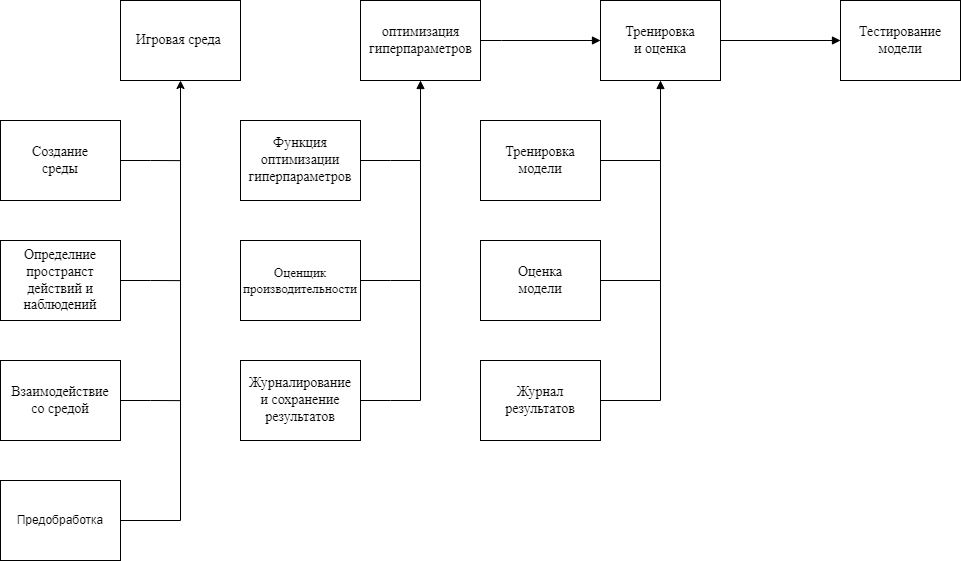


Рисунок 2.1.4 – Схема взаимодействия подсистем, уточненная в соответствии с подсистемами, реализующими тренировку и оценку.

Подсистемы «Тренировка и оценка».

Подсистема «Загрузка обученной модели». Загрузка предварительно обученной модели, которую нужно протестировать.

Подсистема «Цикл тестирования». Цикл, в котором модель взаимодействует со средой на каждом шаге. Модель делает предсказания действий на основе текущего состояния среды и принимает эти действия. После каждого шага обновляется текущее состояние среды и проверяется условие завершения тестирования.

Подсистема «Отображение результатов». Отображение результатов тестирования, таких как полученные награды, состояние среды.

Уточненная схема взаимодействия подсистем представлена на рисунке 2.1.5.



Рисунок 2.1.5 – Схема взаимодействия подсистем, уточненная в соответствии с подсистемами, реализующими тренировку и оценку

Рисунок 2.1.5 представляет собой итоговую схему взаимодействия различных подсистем, задействованных в процессе обучения агента для игры SF2SCE с помощью алгоритма обучения с подкреплением PPO. Схема отражает основные этапы и компоненты процесса обучения агента, включая оптимизацию гиперпараметров, непосредственно процесс обучения с помощью алгоритма PPO, оценку производительности обученной модели, визуализацию и сохранение результатов, а также возможность переподготовки агента с использованием новых данных.

Взаимодействие между различными подсистемами обеспечивает эффективную работу всего процесса обучения и тестирования агента для SF2SCE.

* 1. **ПРОЕКТИРОВАНИЕ ПРОГРАММЫ**

При разработке программы для обучения агента в игре SF2SCE была использована модульная архитектура, основанная на принципах объектно-ориентированного программирования (ООП). Этот подход позволил разбить систему на отдельные модули, каждый из которых отвечает за определенную функциональность, и организовать их взаимодействие с помощью объектов и методов.

**Модульная архитектура:**

Программа была разделена на несколько основных модулей: игровая среда (StreetFighter), обратный вызов для сохранения модели (TrainAndLoggingCallback), обучение агента (PPO), оптимизация гиперпараметров (Optuna) и главный модуль (основной скрипт).

Взаимодействие между модулями осуществляется через объекты и методы. Например, главный модуль создает экземпляры классов StreetFighter, PPO и TrainAndLoggingCallback, а затем вызывает их методы для инициализации, обучения агента и сохранения модели. Такой подход обеспечивает четкое разделение ответственностей между компонентами и упрощает интеграцию и тестирование отдельных модулей.

В целом, использование модульной архитектуры и объектно-ориентированного подхода при разработке программы для обучения агента в игре Street Fighter II обеспечивает модульность, расширяемость, повторное использование кода, простоту интеграции сторонних библиотек и повышенную тестируемость.

**Объектно-ориентированный подход:**

В рамках каждого модуля функциональность была инкапсулирована в классах, следуя принципам ООП.

Классы, такие как StreetFighter, TrainAndLoggingCallback и PPO, представляют собой логические единицы кода, отвечающие за определенные задачи и обладающие собственными атрибутами и методами.

**Взаимодействие между модулями:**

Взаимодействие между модулями осуществлялось через объекты классов и вызовы их методов.

Например, главный модуль создавал экземпляры классов StreetFighter, PPO и TrainAndLoggingCallback, а затем вызывал их методы для инициализации, обучения агента и сохранения модели.

**Использование сторонних библиотек:**

В программе были использованы сторонние библиотеки, такие как retro, stable\_baselines3 и optuna, которые предоставляют готовые классы и функциональность для работы с игровой средой, обучения агента и оптимизации гиперпараметров соответственно.

Эти библиотеки были интегрированы в систему с помощью объектно-ориентированного подхода, создавая экземпляры их классов и взаимодействуя с ними через методы.

Применение модульной архитектуры и объектно-ориентированного подхода при разработке программы принесло следующие преимущества:

* **Модульность**: Разделение системы на отдельные модули упростило разработку, тестирование и сопровождение кода, поскольку каждый модуль мог быть изменен независимо от остальных.
* **Расширяемость**: благодаря инкапсуляции и наследованию, реализованным с помощью ООП, систему можно легко расширять, добавляя новые классы или наследуя функциональность от существующих.
* **Повторное использование кода**: Классы и модули могут быть повторно использованы в других проектах или приложениях, что повышает эффективность разработки.
* **Интеграция сторонних библиотек**: объектно-ориентированный подход упростил интеграцию сторонних библиотек, таких как retro, stable\_baselines3 и optuna, в общую систему.
* **Тестируемость**: модульная структура и инкапсуляция облегчили процесс тестирования отдельных компонентов системы, что повысило качество кода и упростило отладку.

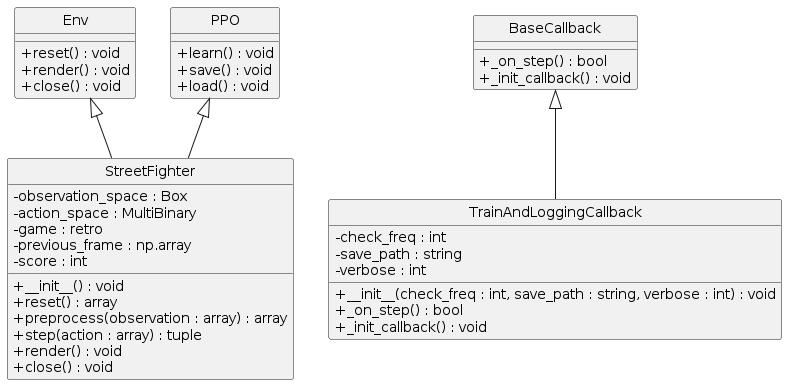
Таким образом, применение модульной архитектуры и принципов объектно-ориентированного программирования при разработке программы для обучения агента в игре SF2SCE обеспечило модульность, расширяемость, повторное использование кода, простоту интеграции сторонних библиотек и повышенную тестируемость. Это способствовало повышению качества, поддерживаемости и масштабируемости системы в целом.

Рисунок 2.2.1 – Диаграмма иерархия классов программы

Рассмотрим основные классы, изображенные на диаграмме 2.2.1:

Env является базовым классом для среды, который не наследуется от других классов в предоставленном контексте, но в библиотеке gym служит абстрактным классом для всех сред. Он определяет основные методы reset, render и close, которые необходимы для работы с любой средой в gym.

PPO представляет собой реализацию алгоритма PPO для обучения моделей глубокого обучения с подкреплением. Этот класс не наследуется непосредственно от других классов в предоставленном контексте, но является частью библиотеки stable\_baselines3 и предоставляет методы learn, save и load для обучения, сохранения и загрузки моделей.

BaseCallback служит базовым классом для функций обратного вызова, который может быть расширен для создания пользовательских функций обратного вызова в процессе обучения. В контексте stable\_baselines3, это базовый класс для всех функций обратного вызова, определяющий методы \_on\_step и \_init\_callback для взаимодействия с процессом обучения.

StreetFighter, наследуя от Env, представляет собой пользовательскую среду для игры SF2SCE. Этот класс расширяет базовый класс Env, предоставляя специфическую реализацию методов reset, step, render и других, адаптированных под игру SF2SCE. Он включает в себя логику для предобработки наблюдений, выполнения действий в среде и получения наград.

TrainAndLoggingCallback, расширяя BaseCallback, предназначен для логирования процесса обучения и сохранения моделей на определённых этапах. Этот класс добавляет специфическую логику для сохранения модели и логирования процесса обучения, определяя методы \_\_init\_\_, \_on\_step и \_init\_callback для настройки частоты проверки, пути сохранения и уровня подробности логирования.

1. **РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ**

**3.1 РЕАЛИЗАЦИЯ ПРОГРАММНЫХ СРЕДСТВ**

Реализация программных средств была направлена на создание эффективной среды для обучения искусственного интеллекта в контексте игры SF2SCE. Основой для этого послужили средства программирования, предоставляющие необходимые инструменты для моделирования среды, обработки данных и взаимодействия с агентом.

**Инициализация среды**

Инициализация среды была выполнена с использованием библиотеки gym, которая предоставляет доступ к разнообразным симулированным средам, в том числе к играм на основе эмулятора retro. Это позволило настроить среду StreetFighter для последующего обучения агента:

import gym

import retro

env=retro.make(game='StreetFighterIISpecialChampionEdition-Genesis')

**Класс StreetFighter**

Для более детальной настройки и адаптации среды был разработан специализированный класс StreetFighter, который наследует функциональность от базового класса Env. В этом классе реализованы методы для управления состояниями игры, выполнения действий и получения наград, а также для предобработки входных данных, что является ключевым для обучения нейронных сетей:

**Метод \_\_init\_\_**

Метод инициализации класса StreetFighter задаёт начальные параметры среды, такие как пространства действий и наблюдений, а также инициализирует переменные, необходимые для работы среды.

class StreetFighter(gym.Env):

def \_\_init\_\_(self):

super(StreetFighter, self).\_\_init\_\_()

# Определение пространства действий

self.action\_space = MultiBinary(12)

# Определение пространства наблюдений

self.observation\_space = Box(low=0, high=255, shape=(84, 84, 1), dtype=np.uint8)

self.game = retro.make(game='StreetFighterIISpecialChampionEdition-Genesis', use\_restricted\_actions=retro.Actions.FILTERED)

**Метод reset**

Метод reset вызывается для сброса среды к начальному состоянию. Это важно для начала нового эпизода обучения. Эпизод — это полный цикл взаимодействия агента с игровой средой, начиная с начального состояния и заканчивая терминальным состоянием, таким как окончание раунда, победа или поражение в игре, или достижение предела по времени или шагам. В каждом эпизоде агент принимает серию решений (действий), основываясь на своей политике, стремясь максимизировать суммарную награду. Метод возвращает начальное наблюдение, которое будет первым входным сигналом для агента.

def reset(self):

obs = self.game.reset()

obs = self.preprocess(obs)

self.previous\_frame = obs

self.score = 0

return obs

**Метод preprocess**

Метод preprocess принимает кадр игры в качестве входных данных и выполняет его предобработку, включая преобразование в оттенки серого и изменение размера.

def preprocess(self, observation):

gray = cv2.cvtColor(observation, cv2.COLOR\_BGR2GRAY)

resize = cv2.resize(gray, (84,84), interpolation=cv2.INTER\_CUBIC)

channels = np.reshape(resize, (84,84,1)

return channels

Изображение игровой среды и игровой среды после процесса предобработки представлены на рисунках 3.1.1. и 3.1.2. соответственно.

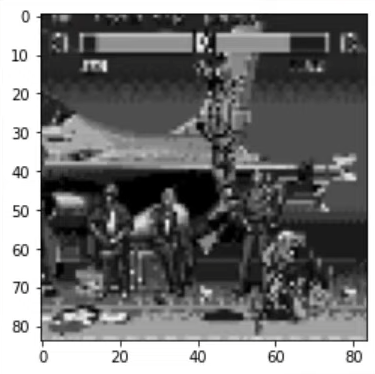
Рисунок 3.1.1 – изображение игровой среды

Рисунок 3.1.2 – изображение игровой среды с пониженным разрешением и сменой цветового канала на черно-белый

**Метод step**

Метод step выполняет действие в среде и возвращает разницу между текущим и предыдущим кадрами, награду, флаг завершения и дополнительную информацию.

def step(self, action):

obs, reward, done, info = self.game.step(action)

obs = self.preprocess(obs)

frame\_delta = obs - self.previous\_frame

self.previous\_frame = obs

reward = info['score'] - self.score

self.score = info['score']

return frame\_delta, reward, done

Изображение разницы между кадрами показаны на рисунке 3.1.3.

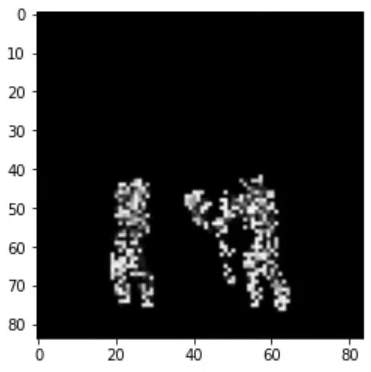
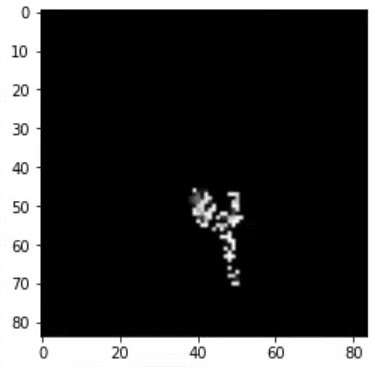


Рисунок 3.1.3 – Изображений дельта-кадров между текущим и предыдущим кадрами

**Метод render**

Метод render отвечает за отображение текущего состояния игры.

def render(self, \*args, \*\*kwargs):

self.game.render()

**Метод close**

Метод close закрывает среду и освобождает ресурсы.

def close(self):

self.game.close()

Эти методы обеспечивают полный цикл работы среды, начиная от инициализации и заканчивая получением и обработкой данных о состоянии игры, что является основой для обучения агента.

**Интеграция среды и агента**

Для успешной интеграции среды StreetFighter, разработанной на основе библиотеки gym, с агентом, использующим алгоритмы глубокого обучения с подкреплением, была применена библиотека stable\_baselines3. Эта библиотека предоставляет реализации современных алгоритмов обучения с подкреплением, включая PPO, которые можно эффективно использовать для обучения агентов в различных средах.

Процесс интеграции начинается с обертывания созданной среды StreetFighter в мониторинг и векторизацию, что позволяет логировать детальную информацию о процессе обучения и обрабатывать данные в формате, подходящем для алгоритма PPO:

env = StreetFighter()

env = Monitor(env, LOG\_DIR)

env = DummyVecEnv([lambda: env])

env = VecFrameStack(env, 4, channels\_order='last')

Эти обертки позволяют логировать детальную информацию о процессе обучения, векторизовать среду для одновременного управления несколькими экземплярами и стекировать кадры для улучшения восприятия временной последовательности агентом.

**Визуализация и мониторинг**

Для визуализации и мониторинга процесса обучения агента были использованы встроенные функции render и matplotlib для отображения состояния игры и создания графических отчетов о производительности агента:

# Визуализация состояния игры с помощью встроенного метода render

env.render()

# Использование matplotlib для отображения обработанного кадра

plt.imshow(cv2.cvtColor(obs, cv2.COLOR\_BGR2RGB))

plt.show()

Кроме того, для логирования процесса обучения и визуализации ключевых метрик в реальном времени был использован TensorBoard, интеграция с которым осуществляется через параметр tensorboard\_log при создании модели PPO:

model = PPO('CnnPolicy', env, tensorboard\_log=LOG\_DIR, verbose=1, \*\*model\_params)

Таким образом, благодаря использованию библиотеки stable\_baselines3 и инструментов визуализации, была обеспечена эффективная интеграция среды и агента, а также возможность детального мониторинга процесса обучения. Это создало устойчивую платформу для разработки, тестирования и оценки производительности агента в динамической среде игры.

**Оптимизация гиперпараметров**

Оптимизация гиперпараметров является критически важным шагом в процессе обучения моделей глубокого обучения с подкреплением. В предоставленном коде используется библиотека Optuna для автоматизации поиска оптимальных гиперпараметров для алгоритма PPO. Рассмотрим, за что отвечает каждый из гиперпараметров, которые оптимизируются в функции optimize\_ppo:

**n\_steps:** Количество шагов сбора данных перед выполнением очередного обновления политики. Большее значение может привести к более стабильному обучению, но увеличивает требования к памяти и может замедлить обучение.

**gamma:** Коэффициент дисконтирования, используемый для расчета будущих наград. Значение близкое к 1 означает, что будущие награды имеют большой вес, в то время как значение ближе к 0 делает агента более "короткозорким".

**learning\_rate:** Скорость обучения, определяющая величину изменений весов нейронной сети во время каждого обновления. Слишком большое значение может привести к нестабильному обучению, а слишком маленькое - к замедлению процесса обучения.

**clip\_range:** Параметр, используемый в PPO для ограничения (клиппинга) отношения вероятностей политик, что предотвращает слишком большие изменения политики и помогает стабилизировать обучение.

**gae\_lambda:** Параметр для расчета Generalized Advantage Estimation (GAE), который используется для уменьшения дисперсии оценки преимущества при сохранении несмещенности. Значение близкое к 1 уменьшает дисперсию, но увеличивает смещение оценки.

Вот как оптимизация гиперпараметров реализуется в коде:

def optimize\_ppo(trial):

return {

'n\_steps': trial.suggest\_int('n\_steps', 2048, 8192),

'gamma': trial.suggest\_loguniform('gamma', 0.8, 0.9999),

'learning\_rate': trial.suggest\_loguniform('learning\_rate', 1e-5, 1e-4),

'clip\_range': trial.suggest\_uniform('clip\_range', 0.1, 0.4),

'gae\_lambda': trial.suggest\_uniform('gae\_lambda', 0.8, 0.99)

}

В этом фрагменте кода trial.suggest\_\* функции используются для определения диапазона значений, которые Optuna будет исследовать. Затем Optuna автоматически запускает серию экспериментов, в каждом из которых она выбирает конкретные значения гиперпараметров и оценивает производительность модели на основе этих значений. Лучшие параметры сохраняются и могут быть использованы для обучения окончательной модели.

**Тренировка и тестирование моделей**

Тренировка и тестирование моделей глубокого обучения с подкреплением являются ключевыми этапами в разработке агентов, способных эффективно взаимодействовать со средой. В предоставленном коде описывается процесс обучения и тестирования модели PPO с использованием пользовательской среды для игры SF2SCE. Рассмотрим подробнее каждый из этих этапов.

Тренировка модели начинается с создания экземпляра среды StreetFighter, который затем оборачивается в несколько слоев для мониторинга, векторизации и стекирования кадров:

env = StreetFighter()

env = Monitor(env, LOG\_DIR)

env = DummyVecEnv([lambda: env])

env = VecFrameStack(env, 4, channels\_order='last')

После этого создаётся модель PPO с определёнными гиперпараметрами, и запускается процесс обучения:

model\_params = study.best\_params # Получение лучших гиперпараметров из процедуры оптимизации

model = PPO('CnnPolicy', env, tensorboard\_log=LOG\_DIR, verbose=1, \*\*model\_params)

model.learn(total\_timesteps=100000, callback=callback)

Во время обучения используется callback, который сохраняет модель через определённые интервалы времени, позволяя вам восстановить процесс обучения с последнего сохранения или использовать промежуточные модели для тестирования.

**Тестирование модели**

После тренировки модель тестируется для оценки её производительности. Это делается с помощью функции evaluate\_policy, которая выполняет несколько эпизодов в среде и вычисляет среднее вознаграждение.

mean\_reward, \_ = evaluate\_policy(model, env, n\_eval\_episodes=1)

Тестирование модели позволяет убедиться, что агент научился выполнять задачи в среде и достигает высоких результатов.

**Экспериментирование с моделью**

После тестирования модели можно провести дополнительные эксперименты, чтобы увидеть, как агент взаимодействует со средой в реальном времени:

obs = env.reset()

done = False

while not done:

action = model.predict(obs)[0]

obs, reward, done, info = env.step(action)

env.render()

time.sleep(0.01)

print(reward)

Этот код позволяет визуализировать, как агент играет в игру, и выводит получаемые награды.

**3.2 ОБУЧЕНИЕ И ТЕСТИРОВАНИЕ МОДЕЛЕЙ**

Стоит также отметить два важных параметра для оценки производительности наших моделей, это:

Средняя длина эпизода (ep\_len\_mean): Этот показатель отражает, насколько долго агент справляется с задачами в среде до завершения эпизода. Модель, показывающая более высокую среднюю длину эпизода, потенциально лучше адаптируется к среде, поскольку способна действовать в ней дольше, прежде чем достигнуть терминального состояния.

Средняя награда за эпизод (ep\_rew\_mean): Этот показатель демонстрирует, насколько эффективно агент максимизирует получаемую награду. Модель с более высокой средней наградой за эпизод более успешно находит стратегии, приводящие к высоким наградам.

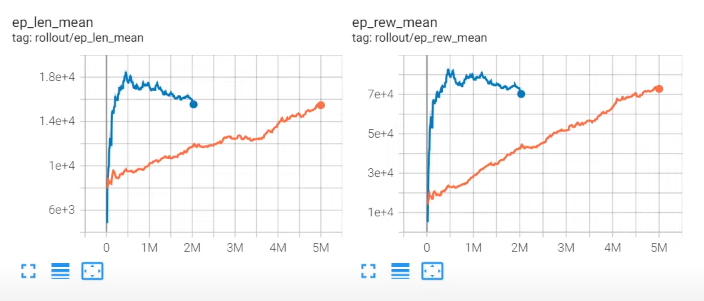
На основе полученных графиков проанализируем как гиперпараметры повлияли на обучение агента. Рассмотрим две самые результативные модели. Оранжевый график – PPO\_1, синий график – PPO\_2.

Рисунок 3.2.1 – Графики средней длины эпизода и средней награды за эпизод для моделей PPO\_1 и PPO\_2.

Модель PPO\_1 имеет следующие гиперпараметры и результаты:

**Награда**: 73368.0

**Learning rate**: 2e-07

**n\_steps**: 2560

**gamma**: 0.906

**gae\_lambda**: 0.891

Для модели PPO\_2 имеем следующие гиперпараметры и результаты:

**Награда**: 70209.0

**Learning rate**: 3e-04

**n\_steps**: 2112

**gamma**: 0.991

**gae\_lambda**: 0.952

Исходя из предоставленных данных, модель PPO\_1 является более эффективной по сравнению с PPO\_2. Модель PPO\_1 достигла высокой общей награды в 73368.0, что свидетельствует о её способности оптимизировать стратегии для максимизации наград в среде. Низкий learning rate (2e-07) обеспечил постепенное и стабильное обучение, позволяя модели избегать значительных колебаний в процессе обучения. Значение n\_steps в 2560 позволило собирать достаточный объем данных для обновления, что улучшило качество обучения. Параметры gamma и gae\_lambda, близкие к 1, указывают на то, что модель уделяла значительное внимание будущим наградам, что способствовало разработке долгосрочных стратегий.

Модель PPO\_2, хотя и показала хорошие результаты с общей наградой 70209.0, уступает оранжевой модели в эффективности. Более высокий learning rate (3e-04) привёл к более быстрым изменениям в весах модели, что вызвало нестабильность в обучении. Меньшее значение n\_steps в 2048 привело к частым обновлениям модели на основе меньшего объема данных, что снизило качество обучения. Высокие значения gamma и gae\_lambda подтверждают стремление модели к учёту будущих наград, однако агрессивный подход к обучению не позволил достичь такой же высокой общей награды, как у модели PPO\_1.

Модель PPO\_1 демонстрирует лучшие результаты благодаря более сбалансированному подходу к обучению, который позволил ей эффективно адаптироваться к среде и максимизировать награды.

В ходе тестирования моделей непосредственно в игровой среде Street Fighter II, обе модели продемонстрировали высокую эффективность в игре против встроенного игрового ИИ. Модель PPO\_1, благодаря тщательно подобранным гиперпараметрам, выиграла 936 из 1000 игр, что свидетельствует о ее превосходной способности к обучению и адаптации к стратегиям противника. Модель PPO\_2 также показала хорошие результаты, выиграв 852 из 1000 игр, но с меньшей частотой побед по сравнению с PPO\_1.

Для большей ясности производительность и гиперпараметры моделей занесем в таблицу 4.1.

Таблица 4.1 – сравнение производительности и гиперпараметров моделей PPO\_1 и PPO\_2:

| **Параметр/Модель** | **PPO\_1** | **PPO\_2** |
| --- | --- | --- |
| Количество игр | 1000 | 1000 |
| Количество побед | 936 | 852 |
| Награда | 73368.0 | 70209.0 |
| Learning Rate | 2e-07 | 3e-04 |
| n\_steps | 2560 | 2112 |
| gamma | 0.906 | 0.991 |
| gae\_lambda | 0.891 | 0.952 |

Эти результаты подчеркивают важность оптимизации гиперпараметров для улучшения производительности моделей в сложных игровых средах. Модель PPO\_1, с ее более низким learning rate и сбалансированными значениями gamma и gae\_lambda, показала, что эти параметры могут быть ключевыми для достижения высокой производительности в игре Street Fighter II.

1. **ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ**
   1. **ИСХОДНЫЕ ДАННЫЕ ДЛЯ РАСЧЕТА ЭКОНОМИЧЕСКОГО ЭФФЕКТА**

Наименование проекта – «Разработка и обучение ИИ для игры SF2SCE с использованием глубокого обучения». В рамках данного дипломного проекта создается программное обеспечение, предназначенное для обучения искусственного интеллекта управлять персонажами в игре Street Fighter II. ПО включает в себя функции для взаимодействия с игровым окружением, обработки и анализа игровых ситуаций, принятия решений и выполнения действий в игре, а также обучения на основе полученных результатов и их визуализации.

Среда разработки ПО – PyCharm 2023.1, язык программирования – Python. Проект включает в себя разработку кастомизированной среды на основе библиотеки gym-retro, предобработку входных данных для уменьшения размерности пространства состояний, а также настройку и оптимизацию гиперпараметров алгоритма PPO для достижения наилучших результатов обучения.

Разработка программного продукта предусматривает выполнение всех ключевых этапов проектирования программного обеспечения, от технического задания до рабочего проекта, и относится к второй группе сложности из-за использования передовых методов машинного обучения и необходимости интеграции с внешним игровым окружением.

Для оценки экономического эффекта от разработки и внедрения программного обеспечения будут выполнены следующие расчеты:

* Определение объема функциональных возможностей программного продукта.
* Расчет полной себестоимости разработки программного обеспечения, включая затраты на исследования, разработку, тестирование и оптимизацию.
* Определение отпускной цены программного продукта и расчет чистой прибыли от его реализации, учитывая потенциальный интерес исследовательских организаций и разработчиков игр к подобным технологиям.

Проект направлен на создание инновационного продукта, который может найти широкое применение в области разработки игр и искусственного интеллекта, предоставляя новые возможности для обучения ИИ в комплексных и динамичных средах.

**4.2 РАСЧЕТ ОБЪЕМА ФУНКЦИЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Общий объем ПО (Vo) определяется исходя из количества и объема функций, реализуемых программой (формула 4.1):

, (4.1)

где Vo– общий объем ПО,

Vi– объем функций ПО,

n – общее число функций.

Расчет общего объема программного обеспечения (количество строк исходного кода) предполагает определение объема по каждой функции. Чаще всего на стадии технико-экономического обоснования проекта невозможно рассчитать объем функций, тогда данный объем может быть получен на основании ориентировочной оценки имеющихся фактических данных по аналогичным проектам, выполненным ранее, или по тем применениям нормативов по каталогу функций.

На основании информации о функциях разрабатываемого ПО по каталогу функций определяется общий объем ПО. В зависимости от организационных и технологических условий, в которых разрабатывается ПО, корректируется объем на основе экспертных оценок.

Уточненный объем ПО (Vу) определяется по формуле 4.2:

, (4.2)

где Vуi – уточненный объем отдельной функции в строках исходного кода

(см. таблицу 4.1).

Таблица 4.1 – Перечень и объем функций программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| **Код функции** | **Наименование (содержание) функции** | **Объем функции строк исходного кода** | |
| **По каталогу (Vi)** | **Уточненный (Vyi)** |
| **1** | **2** | **3** | **4** |
| **1. Ввод, анализ входной информации, генерация кодов и процессор входного языка** | | | |
| 102 | Контроль, предварительная обработка и ввод информации | 280 | 55 |
| **3. Формирование и обработка файлов** | | | |
| 303 | Обработка файлов | 1 050 | 46 |
| 304 | Управление файлами | 5240 | 41 |
| 305 | Формирование файлов | 2130 | 31 |
| **6. Тестирование, проведение тестовых испытаний прикладных программ, вспомогательные программные функции** | | | |
| 601 | Проведение тестовых испытаний прикладных  программ в интерактивном режиме | 3 780 | 36 |
| **7. расчетные задачи, формирование и вывод на внешние носители документов сложной формы и файлом** | | | |
| **1** | **2** | **3** | **4** |
| 707 | Графический вывод результатов | 420 | 33 |
|  | ИТОГО | 12 900 | 242 |

Учитывая информацию, указанную в таблице 4.1, о функциях разрабатываемого программного обеспечения, уточненный объем ПО (Vу) составил 242 строк исходного кода вместо предполагаемого количества строк 6520.

* 1. **РАСЧЕТ ПОЛНОЙ СЕБЕСТОИМОСТИ ПРОГРАММНОГО ПРОДУКТА**

Стоимостная оценка программного средства у разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

* Заработную плату (основную – и дополнительную – ).
* Отчисления на социальные нужды ().
* Материалы и комплектующие изделия ().
* Спецоборудование ().
* Машинное время ().
* Расходы на научные командировки ().
* Прочие прямые расходы ().
* Накладные расходы ().

Затраты на освоение и сопровождение программного средства ( и ).

Полная себестоимость () разработки программного продукта рассчитывается как сумма расходов по всем статьям с учётом рыночной стоимости аналогичных продуктов.

Основной статьёй расходов на создание программного продукта является заработная плата проекта (основная и дополнительная) разработчиков (исполнителей) (), в число которых принято включать инженеров-программистов, руководителей проекта, системных архитекторов, дизайнеров, разработчиков баз данных, Web-мастеров и других специалистов, необходимых для решения специальных задач в команде.

Расчёт заработной платы разработчиков программного продукта начинается с определения:

* Продолжительности времени разработки (), которое устанавливается

экспертным путем с учётом сложности, новизны ПП и фактически

затраченного времени. В данном дипломном проекте  –   60 дней.

* Количества разработчиков ПП. В данном дипломном проекте один разработчик.

Заработная плата разработчиков определятся как сумма основной и дополнительной заработной платы всех исполнителей.

Основная заработная плата каждого исполнителя определяется по формуле (4.3).

, (4.3)

где – тарифная ставка 1-го разряда рабочего (На дату написания дипломного проекта её значение равняется – 250 бел.руб.);

– тарифный коэффициент согласно тарифному разряду исполнителя;

– среднее количество рабочих дней в месяце;

– фонд рабочего времени исполнителя (продолжительность разработки программного модуля, дни);

– коэффициент премии, = 1,5.

Рассчитаем основную заработную плату инженера-программиста и техника-программиста согласно формуле 4.3. Тарифный коэффициент согласно 13 разряду инженера-программиста = 2,17. Продолжительность разработки программного продукта – 60 дней.

*Основная заработная плата инженера-программиста 13 разряда (КПР = 1,5):*

(бел. руб.)

Дополнительная заработная плата каждого исполнителя ( – 20%). Рассчитывается от основной заработной платы по формуле (4.4).

, (4.4)

*Дополнительная заработная плата инженера-программиста 13 разряда:*

(бел. руб.)

Результаты вычислений внесём в таблицу 4.2.

Таблица 4.2 – Расчет заработной платы

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Категория работников | Разряд | Тарифные коэффициент ( | , (дн.) | Коэффициент премии ) | (час.) | Заработная плата, бел. руб. | | |
| Основная | Дополнительная | Всего |
| Инженер-программист | 113 | 2,17 | 44 | 1,5 | 8 | 1484,2 | 296,8 | 1781 |
| Итого | - | - | - | - | - | 1484,2 | 296,8 | 1781 |

Таким образом, как видно из таблицы, заработная плата инженера- программиста составляет 1953 бел. руб.

Отчисления на социальные нужды () определяются по формуле (4.5) в соответствии с действующим законодательством по нормативу (35% – отчисления в ФСЗН):

, (4.5)

(бел. руб.)

Расходы по статье «Спецоборудование» () включает затраты на приобретение технических и программных средств специального назначения, необходимых для разработки методического пособия, включая расходы на проектирование, изготовление, отладку и другое.

В данном дипломном проекте для разработки алгоритма для анализа эмоций на основе сигналов электроэнцефалографии на базе Python, TensorFlow приобретение какого-либо спецоборудования не предусматривалось. Так как спецоборудование не было приобретено, данная статья не рассчитывается.

По статье «Материалы и комплектующие изделия» () отражаются расходы на бумагу, картридж и красящие ленты для принтера, необходимые для разработки ПП. Норма расхода материалов в суммарном выражении определяются в расчете на 100 строк исходного кода. В данном дипломном проекте не рассчитывается.

Расходы по статье «Машинное время» () включают оплату машинного времени, необходимого для разработки и отладки программного продукта. Они определяются в машино-часах по нормативам на 100 строк исходного кода машинного времени.  определяется по формуле (4.6).

, (4.6)

где – цена одного машинного часа (1,6 бел. руб.);

– уточнённый общий объём функций строк исходного кода (LOC);

– норматив расхода машинного времени на отладку 100 строк кода, машино-часов. Принимается в размере 0,8.

(бел. руб.)

Расходы по статье «Научные командировки» () берутся либо по смете научных командировок, разрабатываемой на предприятии, либо в процентах от основной заработной платы исполнителей (10-15%). Так как в данном проекте научные командировки не предусмотрены, данная статься не рассчитывается.

Расходы по статье «Прочие затраты» () включают затраты на приобретение специальной научно-технической информации и специальной литературы и определяются по формуле (4.7).

(4.7)

Где:

НПЗ – норматив прочих затрат, НПЗ = 15%.

(бел. руб.)

Затраты по статье «Накладные расходы» () связаны с содержанием вспомогательных хозяйств, а также с расходами на общехозяйственные нужды. Определяется по нормативу в процентах к основной заработной плате по формуле (4.8).

, (4.8)

где – норматив накладных расходов, в данном дипломном проекте норматив накладных расходов равен 40%.

(бел. руб.)

Сумма вышеперечисленных расходов по статьям на программный продукт служит исходной базой для расчёта затрат на освоение и сопровождение программного продукта. Они рассчитываются по формуле (4.9).

, (4.9)

(бел. руб.)

Организация-разработчик участвует в освоении программного продукта и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Затраты на освоение определяются по установленному нормативу от суммы затрат по формуле (4.10).

, (4.10)

где – установленный норматив затрат на освоение. Для данного дипломного проекта принимается равной 5%.

176,74 (бел. руб.)

Организация-разработчик осуществляет сопровождение программного продукта и несёт расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Эти расходы рассчитываются по формуле (4.11).

, (4.11)

где – установленный норматив затрат на сопровождение программного продукта. Для данного дипломного проекта принимается равной̆ 5%.

(бел. руб.)

Полная себестоимость () разработки программного продукта рассчитывается как сумма расходов по всем статьям. Она определяется по формуле (4.12).

, (4.12)

(бел. руб.)

Результаты вычислений занесём в таблицу 4.3.

Таблица 4.3 – Себестоимость программного продукта

|  |  |  |
| --- | --- | --- |
| **Наименование статей затрат** | **Норматив, %** | **Сумма затрат, бел. руб.** |
| Заработная плата, всего | - | 1 953 |
| Основная заработная плата | - | 1627,5 |
| Дополнительная заработная плата | - | 325,5 |
| Отчисления на специальные нужды | 35% |  |
| Спецоборудование | Не применялось | - |
| Материалы | Не применялись | - |
| Машинное время | - | 3,1 |
| Научные командировки | Не планировались | - |
| Прочие затраты | 15% | 244,13 |
| Накладные расходы | 40% | 651 |
| Сумма затрат | - |  |
| Затраты на освоение | 5% |  |
| Затраты на сопровождение | 5% |  |
| Полная себестоимость | - |  |

В результате всех расчётов полная себестоимость ПП составила бел. руб.

## 4.4 РАСЧЕТ ЦЕНЫ И ПРИБЫЛИ ПО ПРОГРАММНОМУ ПРОДУКТУ

Для определения цены программного продукта необходимо рассчитать плановую прибыль, которая рассчитывается по формуле (4.13).

, (4.13)

где:

– полная себестоимость программного модуля, бел. руб;

R – уровень рентабельности программного модуля. В данном дипломном проекте уровень рентабельности равен 20%.

(бел. руб.)

После расчета прибыли от реализации по формуле (4.14) определяется прогнозируемая цена программного продукта без налогов.

, (4.14)

(бел. руб.)

Отпускная цена (цена реализации) программного продукта включает налог на добавленную стоимость и рассчитывается по формуле (4.15).

, (4.15)

где:

– налог на добавленную стоимость.

Для данного программного продукта рассчитывается по формуле (4.16).

, (4.16)

где:

– прогнозируемая цена, бел. руб.;

НДС – налог на добавленную стоимость. В настоящее время – это 20%.

(бел. руб.)

(бел. руб.)

Прибыль от реализации программного продукта за вычетом налога на прибыль является чистой прибылью (). Чистая прибыль остаётся организации-разработчику и представляет собой экономический эффект от создания нового программного продукта. Она рассчитывается по формуле (4.17).

, (4.17)

где – ставка налога на прибыль. В настоящее время он равен 20%.

(бел. руб.)

Все расчёты цены и прибыли по ПП сведены в таблицу 4.4.

Таблица 4.4 – Расчёт отпускной цены и чистой прибыли программного модуля

|  |  |  |
| --- | --- | --- |
| **Наименование статей затрат** | **Норматив, %** | **Сумма затрат, бел. руб.** |
| Полная себестоимость | - |  |
| Прибыль | 20 |  |
| Цена без НДС | - |  |
| НДС | 20 |  |
| Отпускная цена | - |  |
| Налог на прибыль | 20 | 155,53 |
| Чистая прибыль | - |  |

В ходе произведенных расчетов определены основные экономические показатели:

* Полная себестоимость – бел. руб.
* Прогнозируемая цена – 5599,09 бел. руб.
* Чистая прибыль – 622,12 бел. руб.

Разработанный программный продукт имеет малое количество конкурентов с более высокими ценами на их услуги. Таким образом, рассчитанная отпускная цена на программный продукт, разрабатываемой в рамках данного дипломного проекта, является конкурентоспособной. При расчете цены учтены отчисления в фонд социальной защиты, а также налоги, необходимые к уплате. К конечному итогу получаем окончательную цену продукта, равную 5599,09 белорусских рубля.

# 5. ЭНЕРГОСБЕРЕЖЕНИЕ

Уменьшение количества потребляемой энергии становится одной из важнейших проблем современности. Сегодня на энергосбережение направлено пристальное внимание мировой общественности. С появлением высокоиндустриального общества, опасное вмешательство человека в природу резко усилилось, расширился объём этого вмешательства, оно стало многообразнее и на современном этапе грозит стать глобальной опасностью для человечества. Расход невозобновляемых видов сырья повышается пропорционально росту количества фабрик и заводов. Ежедневно на обеспечение человечества достаточным количеством топлива и энергии направляются максимальные усилия. Ограниченность запасов топливно-энергетических ресурсов заставляет современного человека обратиться к энергосбережению как одному из основных элементов концепции развития мировой энергетики.

Энергосбережение означает рациональное использование энергоресурсов во всех звеньях преобразования энергии - от добычи первичных энергоресурсов до потребления всех видов энергии конечными пользователями.

На развитие хозяйствующих субъектов в нашей стране существенное негативное влияние оказывает высокая доля энергетических затрат в издержках производства, которая на промышленных предприятиях составляет в среднем 8-12% и имеет устойчивую тенденцию к росту в связи с большим моральным и физическим износом основного оборудования и значительными потерями при транспортировке энергетических ресурсов. Одним из определяющих условий снижения издержек на промышленных предприятиях и повышения экономической эффективности производства в целом является рациональное использование энергетических ресурсов.

Энергосберегающий путь развития экономики возможен только при формировании и последующей реализации программ энергосбережения на отдельных предприятиях. Уменьшение потребления энергии позволяет без ущерба для производства повысить удельный вес собственных энергоресурсов в общем количестве потребляемой энергии.

Координацию работ в этом направлении и осуществление надзорных функций осуществляет Комитет по энергоэффективности Республики Беларусь и его территориальные областные управления. Под руководством комитета разработана Республиканская программа по энергосбережению, которая является фундаментом для внедрения методов эффективного и рационального использования энергии во всех сферах деятельности, связанной с производством, передачей, распределением и потреблением энергии.

Энергосбережение (экономия электроэнергии) – реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на рациональное использование и экономное расходование топливно-энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии. Энергосбережение – важная задача по сохранению природных ресурсов.

Энергоэффективность – эффективное использование энергетических ресурсов – достижение экономически оправданной эффективности использования технико-экономических ресурсов с модернизацией существующей техники и технологии и при соблюдении требований к охране окружающей среды.

Более 30 миллиардов киловатт-часов энергии ежегодно тратятся впустую только из-за того, что люди ленятся вовремя выключить свой компьютер. Если бы к данной проблеме относились серьезнее, то, кроме всего прочего, это помогло бы сэкономить более 3 млрд долларов. Выбросы СО2 от сжигания топлива для генерации электроэнергии для питания всего 15 среднестатистических компьютеров равны таковым от одного автомобиля.

Из одного барреля нефти (160 литров) вырабатывается около 556 кВт•ч электроэнергии. Мощность, потребляемая современным системным блоком, в среднем достигает 200 Вт. Добавьте к этому еще 80 Вт, если используется ЭЛТ-монитор (значение для жидкокристаллических моделей примерно в два раза меньше). Таким образом, среднестатистический ПК нуждается примерно в 1 кВт каждые 4 часа. Если оставить компьютер работать в режиме 24/7 (а ведь именно так многие пользователи и поступают), он будет «съедать» один баррель нефти каждые 90 дней. При оптимизации уровня его энергопотребления можно увеличить длительность данного периода до 6 месяцев и более. Каждый использованный на питание ПК киловатт электроэнергии условно равен попаданию в атмосферу порядка 0,68 кг CO2, что за день составит примерно 8 кг, за год 1,49 тонны CO2, и это всего от одного компьютера!

Из-за удорожания электроэнергии и ужесточения экологических стандартов на энергетическое и радиационное загрязнение среды на самых больших рынках развитых стран, сегодня все производители переходят на технологии уменьшения потребления электроэнергии и ПК, и периферией, особенно в моменты простоя и ожидания. Это касается и мониторов, и принтеров, и всяческой бытовой техники.

**Энергосбережение в центрах обработки данных (ЦОД) или дата-центрах**

По подсчетам экспертов, на строительство и дальнейшую эксплуатацию крупного дата-центра в течение пяти лет может уходить до 15 и более миллионов долларов. Расходы на оплату электроэнергии обычно занимают вторую строку. По разным оценкам, это 20-25%.

Существует ряд мер, которые помогают уменьшит энергозатраты:

* Применение эффективных систем кондиционирования и охлаждения ЦОД, например, систем свободного охлаждения. Они ощутимо дороже стандартных решений и имеют большие габариты, но за счет «холодных» месяцев (до полугода) в разы снижают потребление электроэнергии, а потому довольно быстро окупаются.
* Оптимизация количества единиц техники. Это снижает потребляемое электричество и уменьшает количество выделяемого тепла, что напрямую связано с инженерной инфраструктурой, как наиболее энергоемкой и критичной системой для обеспечения бесперебойной работы всего оборудования в ЦОД.
* Перенос ЦОД из мегаполисов в регионы.
* Применение технологии виртуализации. Данная технология позволяет значительно снижать затраты на оборудование и ПО ЦОД, повышать коэффициент загрузки серверов, получать большую степень доступности, а также существенно уменьшает энергопотребление.
* Использование солнечных батарей в качестве дополнительного и экологически чистого источника энергии.

**Энергосбережение на серверах**

Blade-серверы – это ближайшее будущее традиционных серверных технологий. С одной стороны, blade-серверы - альтернатива решениям на базе разрозненных серверов начального и среднего уровня, с другой - дорогостоящим многопроцессорным серверам с симметричной архитектурой. Как правило, blade-серверы отличаются высокой эффективностью энергосбережения.

В новых системах используются новейшие микропроцессоры AMD и Intel с низким напряжением питания, а также реализована инновационная технология хранения для blade-серверов на базе устройств флэш-памяти с низким энергопотреблением.

Практически все Blade-решения имеют собственные интеллектуальные системы управления охлаждением и питанием, что позволяет оптимизировать затраты на них в зависимости от потребностей. Помимо значительной экономии электроэнергии (до 20%), ограничения тепловыделения и потребности в системе охлаждения (т.е. сокращения долгосрочных расходов на эксплуатацию), данный подход существенно экономит время IT-персонала и, таким образом, снижает совокупную стоимость владения.

В последние годы такие производители как HP, IBM, Sun, Intel, AMD и другие интенсивно вкладывают деньги в использование энергосберегающих технологий, что вместе с технологией виртуализации серверов дает ощутимую экономию электроэнергии.

Виртуализация серверов – возможность одному компьютеру выполнять работу нескольких. Выигрыш достигается за счет более эффективного использования вычислительных ресурсов; уменьшения количества оборудования при увеличении производительности; снижения расходов на поддержку ИТ; снижение энергозатрат.

По данным IBM, при стандартном классическом использовании серверов, эффективность использования процессоров у мэйнфрэйм (mainframe) составляет около 55%, Unix ~ 25%, x86 ~ 10%. В случае x86, при увеличении числа процессоров, доля времени использования каждого процессора становится еще меньше. Виртуализация и новые аппаратные решения позволяют повысить загрузку x86 серверов до 50-70%.

**Энергосбережение ПК**

Один из наиболее простых и эффективных способов уменьшить энергопотребление ПК – модернизировать его отдельные компоненты или систему в целом. Конечно, процесс затратен в финансовом плане, но бесспорное преимущество для пользователя – зачастую получение высшего уровня быстродействия. Так, процессоры Intel Core 2 Duo гораздо экономичнее, чем памятные Pentium D 8xx/9xx, которые с успехом могли заменить небольшие печки. При этом экономичность современных моделей на порядок выше. Еще один хороший пример – графические адаптеры. Ни для кого не секрет, что некогда топовый Radeon HD 2900 XT отличался хорошим «аппетитом» в плане энергопотребления. Современные видеокарты (к примеру, HD 4850), чипы которых выполнены по более тонкому техпроцессу, наряду с увеличением производительности позволяют сэкономить пару десятков ватт при работе системы.

Мощная современная видеокарта под полной нагрузкой требует столько же энергии, сколько остальные комплектующие ПК вместе взятые: от 110 до 270 Вт. Поэтому производители приступили к выпуску интеллектуальных видеокарт с управлением потребления электроэнергии в зависимости от нагрузки

Все вышесказанное касается и мониторов. Так, потребление устаревших ЭЛТ-моделей находится на отметке 70–100 Вт и более в зависимости от диагонали экрана. Жидкокристаллические обходят их на порядок в плане экономичности – им для работы достаточно 30–50 Вт. Логично: с увеличением диагонали дисплея растет и его энергопотребление. И если переход от 17-дюймового дисплея к 22-дюймовому почти незаметен (30 Вт против 40–45 у ЖК-версий), то уже 30-дюймовые экземпляры потребляют свыше 130 Вт электроэнергии.

Еще один вариант экономии – отказ от слишком мощных блоков питания, которые при обеспечении слабых и средних систем работают с низким КПД. Чем ближе показатели реальной потребляемой мощности от заявленной, тем коэффициент полезного действия выше.

Соблюдая несложные правила, можно значительно сократить энергопотребление своего ПК.

**ЗАКЛЮЧЕНИЕ**

В ходе данной работы была продемонстрирована способность алгоритма Proximal Policy Optimization (PPO) обучать модели, которые успешно сражаются с искусственным интеллектом в игре Street Fighter II. Сравнение различных обученных моделей позволило выявить ключевые гиперпараметры, влияющие на успешность обучения. Модель с оптимально подобранными параметрами learning rate, gamma и gae\_lambda показала превосходные результаты, что подчеркивает важность тщательного подхода к настройке процесса обучения.

Разработанные модели могут быть использованы в качестве основы для дальнейших исследований в области искусственного интеллекта и обучения с подкреплением. Результаты работы открывают перспективы для создания более сложных и эффективных систем ИИ, которые могут найти применение в различных сферах, требующих автономного принятия решений. В случае модификации, представленный алгоритм обучения может быть адаптирован для тренировки на основе действий реального игрока. Это обеспечит его способность поддерживать уровень, соответствующий уровню игрока, тем самым всегда предоставляя достойную конкуренцию.

В целом, данная работа подтверждает потенциал методов глубокого обучения с подкреплением для создания высокопроизводительных агентов, способных адаптироваться к сложным условиям и достигать значительных успехов в конкурентной среде. Это становится особенно важным в контексте развития игровых технологий и стремления к созданию более реалистичных и вызывающих ИИ-противников.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. ГОСТ 7.1-2003. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – Минск, 2004. – 48 с.
2. ГОСТ ЕСПД 19.103-77. Обозначение программ и программных документов.
3. ГОСТ ЕСПД 19.105-78. Общие требования к программным документам.
4. ГОСТ ЕСПД 19.301-2000. Программа и методика испытаний.
5. ГОСТ ЕСПД 19.401-78. Текст программы.
6. ГОСТ ЕСПД 19.402-78. Описание программы.
7. ГОСТ ЕСПД 19.502-78. Описание применения.
8. ГОСТ ЕСПД 19.504-79. Руководство программиста.
9. ГОСТ ЕСПД 19.505-79. Руководство оператора.
10. ГОСТ ЕСПД 19.508-79. Руководство по техническому обслуживанию.
11. ГОСТ ЕСПД 19.701-90. Схемы алгоритмов, программ, данных и систем.
12. NumPy // NumPy [Электронный ресурс]. – Режим доступа: https://numpy.org. –Дата доступа: 02.04.2024.
13. PyCharm: IDE для профессиональной разработки // JetBrains [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/ru-ru/pycharm. – Дата доступа: 12.05.2024.
14. TensorFlow // TensorFlow [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org. – Дата доступа: 07.04.2024.
15. Welcome to Python.org // Python [Электронный ресурс]. – Режим доступа: https://www.python.org. – Дата доступа: 02.04.2024.
16. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347. <https://arxiv.org/abs/1707.06347>. – Дата доступа: 02.04.2024.
17. OpenAI. (n.d.). Gym: A toolkit for developing and comparing reinforcement learning algorithms. <https://gym.openai.com/>. – Дата доступа: 02.04.2024.
18. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv:1606.01540. <https://arxiv.org/abs/1606.01540>. – Дата доступа: 02.04.2024.
19. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. <https://www.nature.com/articles/nature14236>. – Дата доступа: 02.04.2024.
20. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press. <http://incompleteideas.net/book/the-book-2nd.html>. – Дата доступа: 02.04.2024.
21. OpenAI. (n.d.). Stable Baselines3: A reliable implementation of reinforcement learning algorithms. <https://github.com/DLR-RM/stable-baselines3>. – Дата доступа: 02.04.2024.
22. Optuna: A hyperparameter optimization framework. (n.d.). <https://optuna.org/>. – Дата доступа: 02.04.2024.

**СПИСОК СОКРАЩЕНИЙ**

API – Application Programming Interface.

CPU – Central processing unit.

GPU – Graphics processing unit.

IDE – Интегрированная среда разработки.

LOC – Lines Of Code.

RL – Reinforcement learning

PPO – Proximal Policy Optimization

SF2SCE – Street Fighter II': Special Champion Edition