СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время игровая индустрия стала одной из самых быстроразвивающихся отраслей в мире. Кроме того, особое внимание в индустрии уделяется VFX – визуальным эффектам. Визуальные эффекты имеют огромное влияние на итоговый вид произведения и восприятие зрителя и играют одну из решающих ролей в конечном визуале игры.

Одно из направлений темы визуальных эффектов являются системы частиц. Физика частиц часто востребована для улучшения визуальных эффектов или создания того, что невозможно сделать иными способами. В настоящее время системы частиц используются в многочисленных цифровых приложениях, таких как видеоигры, анимация, фильмы и все виды цифрового искусства. Почти в каждой из этих сред в той или иной степени могут использоваться эффекты частиц. Частицы можно использовать для имитации природных явлений, взрывов, пожаров, добавления дождя или дыма в сцену, имитации ветра или разнообразных магических заклинаний. Это лишь некоторые примеры, для которых могут использоваться частицы. [1] [2]

Их легко применять, но сложно разрабатывать, обычно это требует больших временных и бюджетных затрат. Поэтому одна из важных задач в этой области — это задача оптимизации процесса разработки VFX.

В последние годы машинное обучение стало популярным инструментом для решения различных сложных задач связанных с оптимизацией процессов разработки. И с помощью машинного обучения можно добиться улучшения и оптимизации работы пайплайна разработки визуальных эффектов для видеоигр.

Одним из способов использования машинного обучения является использование нейронных сетей. Нейронные сети — это разновидность алгоритма машинного обучения, который основан на том, как работает человеческий мозг. Они состоят из слоев взаимосвязанных "нейронов", которые могут обрабатывать и передавать информацию. В контексте

эффектов нейронные сети могут быть обучены распознавать определенные паттерны или особенности в их структуре и связях параметров.

Машинное обучение потенциально может повысить качество визуальных эффектов видеоигр, и скорость разработки визуальных эффектов. Тем самым удешевить производство и сделать разработку видеоигр доступнее для начинающих разработчиков.

Темой данной выпускной квалификационной работы является разработка метода внедрения машинного обучения в пайплайн создания VFX ассетов для игровых движков.

Цель выпускной квалификационной работы: упростить создание и поиск VFX для игр с помощью генерации ассетов с использованием генеративно – состязательных нейронных сетей.

Для достижения указанной цели в данной работе необходимо решить следующие задачи:

– Анализ предметной области генеративно – состязательных нейронных сетей,

– Разработка теоретического решения инструмента для генерации ассетов партиклей для игровых движков,

– Практическая реализация инструмента,

– Реализация нейронной сети для генерации VFX ассетов,

– Проведение экспериментов с обучением нейронной сети.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

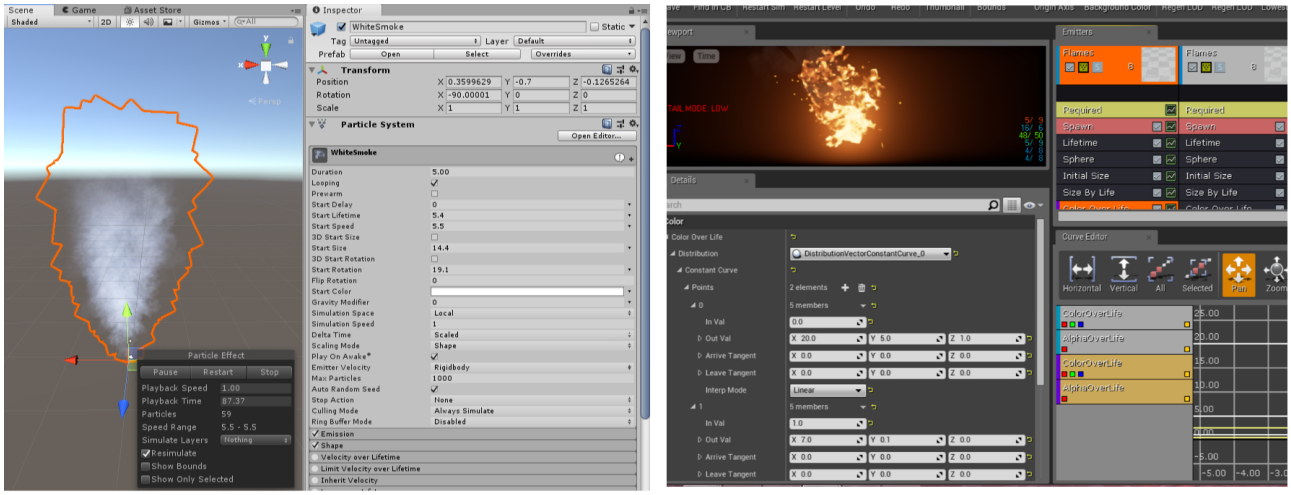
1.1 Введение в VFX и системы частиц

Моделирование таких явлений, как облака, дым, вода и огонь, затруднительны при использовании стандартных подходов графики. Эти "размытые" объекты не имеют гладких, четких поверхностей. Вместо этого они имеют динамические и плавные изменения формы и внешнего вида. Представление систем частиц отличается тремя основными идеями, которые помогают смоделировать эти явления. Во-первых, объект представлен не набором примитивных элементов поверхности, таких как полигоны или пятна, которые определяют его границу, а в виде облаков примитивных частиц, которые определяют его объем. Во-вторых, система частиц не является статичной сущностью. Ее частицы меняют форму и перемещаются с течением времени. Новые частицы "рождаются", а старые "умирают". В-третьих, объект, представленный системой частиц, не является детерминированным, поскольку его форма и вид не заданы полностью. Вместо этого для создания и изменения объекта используются стохастические процессы используются для создания и изменения формы и внешнего вида объекта [3 – 5].

Системы частиц часто основаны на многих параметрах, таких как эмиссия, время жизни, размер и скорость, которые контролируют поведения каждой частицы в отдельности. Совокупность параметров может имитировать различные явления для создания VFX для симуляторов, игр и фильмов. Имитируемые частицы заменяются текстурами billboard – 2D-текстурами спрайтов, обращенными к виртуальной камере, чтобы передать их внешний вид. Также возможно использовать 3D-модели для внешнего вида частиц.

VFX определяет свое поведение на основе параметров/модулей, определенных с плавающими значениями, интерполированными кривыми или случайными значениями между указанными двумя плавающими

значениями. Примерами инструментов для работы с параметрами и системами частиц являются игровые движки и различные 3D пакеты: Blender [6], Unity [7], Unreal Engine [8], они изображены на Рисунок 1.



а) Unity

б) Unreal Engine

Рисунок 1 – Примеры инструментов для работы с системами частиц

1.2 Введение в нейронные сети

Нейронные сети – это тип алгоритма машинного обучения, вдохновленный структурой и функциями человеческого мозга. Они состоят из слоев взаимосвязанных "нейронов", которые обрабатывают и передают информацию. Нейронные сети способны изучать закономерности и взаимосвязи в данных и могут использоваться для широкого спектра задач. В ходе обучения нейронной сети предъявляется большое количество примеров входа/выхода, и она настраивает свои внутренние параметры, чтобы минимизировать ошибку между своими прогнозами и правильными выходами.

Существуют разные типы нейронных сетей, включая сети с прямой связью, свёрточные нейронные сети (CNN) [9] и рекуррентные нейронные сети (RNN) [10].

Генеративные состязательные сети (GAN) [11] – это тип нейронных сетей, используемых для генерации синтетических данных, похожих на заданный набор обучающих данных. Они состоят из двух сетей: сети-генератора, которая генерирует синтетические данные, и сети-дискриминатора, которая пытается отличить синтетические данные от реальных. Обе сети обучаются одновременно, при этом сеть-генератор пытается создать данные, которые могут обмануть сеть-дискриминатор, а сеть-дискриминатор пытается точно отличить синтетические данные от реальных. Благодаря состязанию двух сетей генератора и дискриминатора, по итогу обучения генератор может создавать новые данные, которые могут быть использованы человеком. GAN использовались для решения таких задач, как создание реалистичных изображений и синтез звука. Например, в этой работе [12] разработчики научили сеть генерировать иконки, которые можно было бы использовать как ассеты для игры. Результаты работы сети изображены на Рисунок 2.

Рисунок 2 – Пример работы GAN. Сверху – обучающие данные, снизу – сгенерированные генератором

Рекуррентные нейронные сети (RNN) – это тип нейронных сетей, предназначенных для обработки последовательных данных, таких как естественный язык. Они состоят из блоков, которые могут поддерживать внутреннее состояние и "запоминать" информацию с предыдущих этапов последовательности. Это позволяет RNN улавливать долгосрочные зависимости в данных и выполнять такие задачи, как перевод языка и языковое моделирование.

Автоэнкодеры [13] [14] – состоят из двух частей: кодера, который отображает входные данные в скрытое представление более низкой размерности, и декодера, который отображает скрытое представление

обратно в исходную размерность. Целью автоэнкодера является обучение компактному представлению входных данных, которое отражает наиболее важные особенности или закономерности. Автоэнкодеры могут использоваться для решения таких задач, как уменьшение размерности данных, и выявление признаков. Архитектура автоэнкодера изображена на Рисунок 3

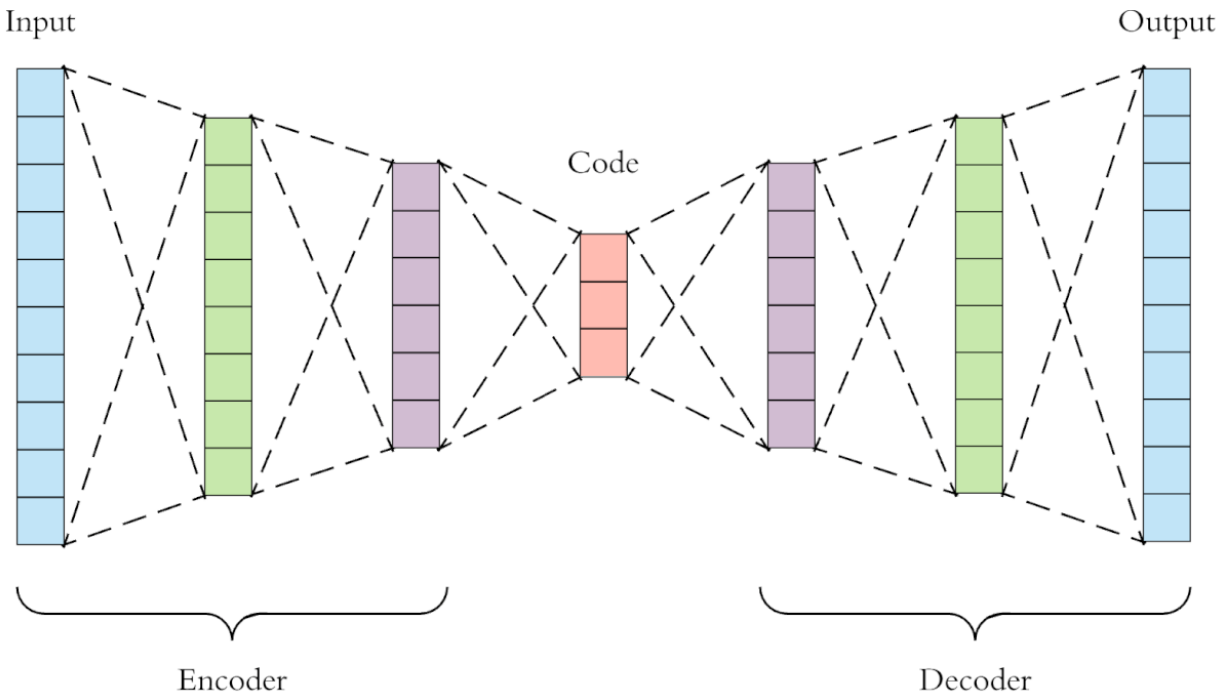


Рисунок 3 – Пример архитектуры Автоэнкодера

1.3 Примеры использования GAN

При обзоре статей были рассмотрены существующие решения и исследования в области применения GAN.

Ambient occlusion generative adversarial network (AOGAN). Разработчики работы – ambient occlusion generative adversarial network (AOGAN) [15] использовали архитектуру нейронной сети – условную генеративно-состязательную сеть (cGAN) для апроксимации эффекта пост процессинга ambient occlusion. Примеры работы AOGAN и сравнение с другими методами изображена Рисунок 4.

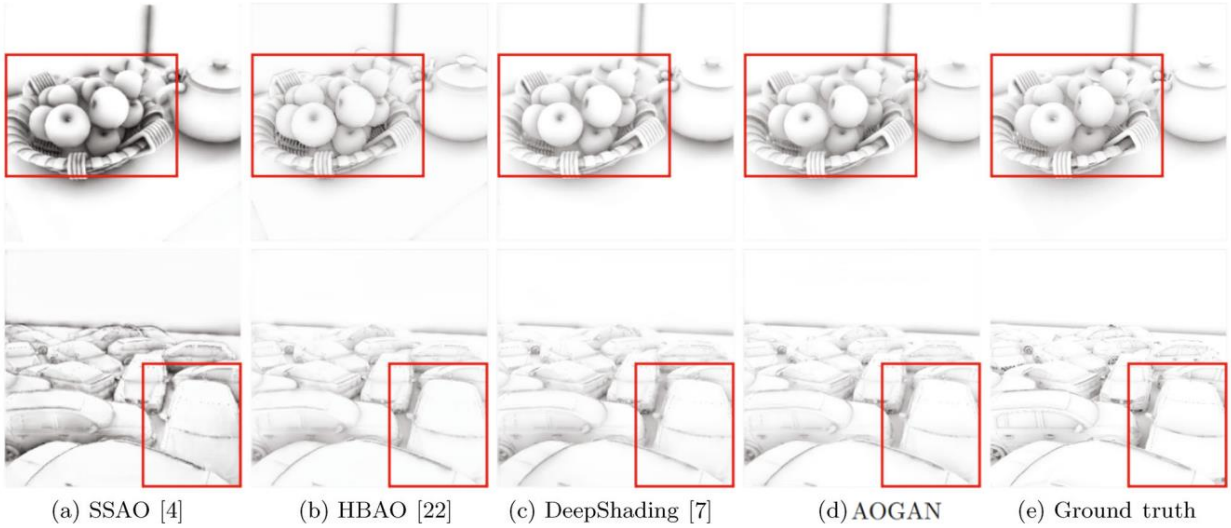


Рисунок 4 – Сравнение результатов работы AOGAN с другими методами. Screen Space Ambient Occlusion (SSAO), Horizon Based Ambient Occlusion (HBAO) и Deep Shading (DS)

AOGAN состоит из генеративной сети (G) и дискриминантной сети (D), которые обучены оптимизировать разработанную функцию потерь. Для функции потерь разработчики используют совмещение нескольких разных функций потерь:

– Потеря состязательная. Результат работы дискриминатора D (сфабрикованное или реальное изображение).

– Потеря восприятия. В этом параметре сравнивают результаты выявления признаков\характеристик с использованием VGG для сгенерированной картинки и эталонной.

– Параметр содержания. Потеря содержимого рассчитывается с использованием показателя индекса структурного сходства (SSIM) между выводом G и эталонным изображением с эффектом.

Сеть генератора использует структуру, основанную на сети UNet [16], дискриминатор использует структуру на основе PatchGan [17]. Также в работе применили механизм внимания [18] который значительно улучшил

результат на маленьких и сложных сценах. Архитектура подхода изображена на Рисунок 5.

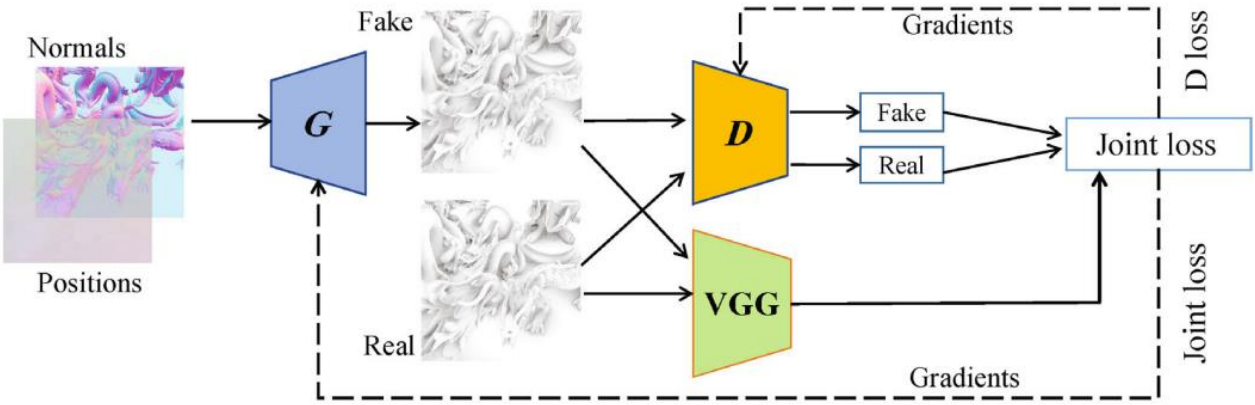


Рисунок 5 – Архитектура AOGAN

Результатом этого подхода стало улучшения качества эффекта что замерялось в потерях по сравнению с оффлайн методом ambient occlusion. Но значительно улучшить производительность не удалось, результаты сравнения представлены на Рисунок 6.

Table 1 Numerical evaluation of various baseline methods. Best scores are highlighted in bold

ID	Method	SSIM	PSNR	Perceptual	Run-time (ms)
A1	SSAO	0.695	17.68	0.310	14.4
A2	HBAO	0.751	19.38	0.186	**8.5**
A3	DS	0.803	21.85	0.231	12.5
A4	Ours	**0.864**	**24.80**	**0.083**	13.5

Рисунок 6 – Сравнение эффективности работы AOGAN с другими методами. Screen Space Ambient Occlusion (SSAO), Horizon Based Ambient Occlusion (HBAO) и Deep Shading (DS)

Имитация голоса с помощью генеративно-состязательных сетей. В статье "Voice Impersonation using Generative Adversarial Networks" [19] рассматривается создание нейронной сети для имитации голоса диктора. Этот метод использует Генеративно-Состязательные сети чтобы анализировать особенности голоса через спектрографические изображения, что позволяет создавать реалистичные голосовые записи.

Авторы показывают, как GAN может успешно передавать уникальные особенности голоса. Разработанная ими модель, VoiceGAN, способна преобразовывать голос одного диктора в голос другого, при этом сохраняя языковые особенности и индивидуальный стиль. Архитектура VoiceGAN изображена на Рисунок 7.

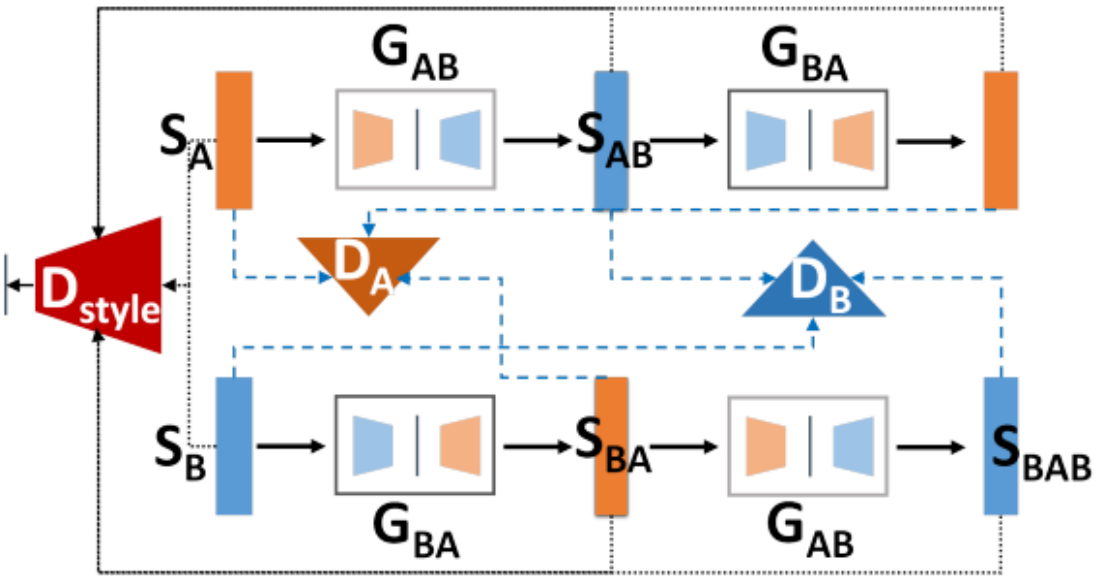


Рисунок 7 – Архитектура VoiceGAN

Модель учитывает различия в длине звуковых сигналов и уникальные стилистические черты голоса, которые сложно измерить. Тестирование на данных TIDIGITS показало, что модель хорошо передает гендерные характеристики голоса. Пример генерации гендерных особенностей голоса изображен на Рисунок 8.

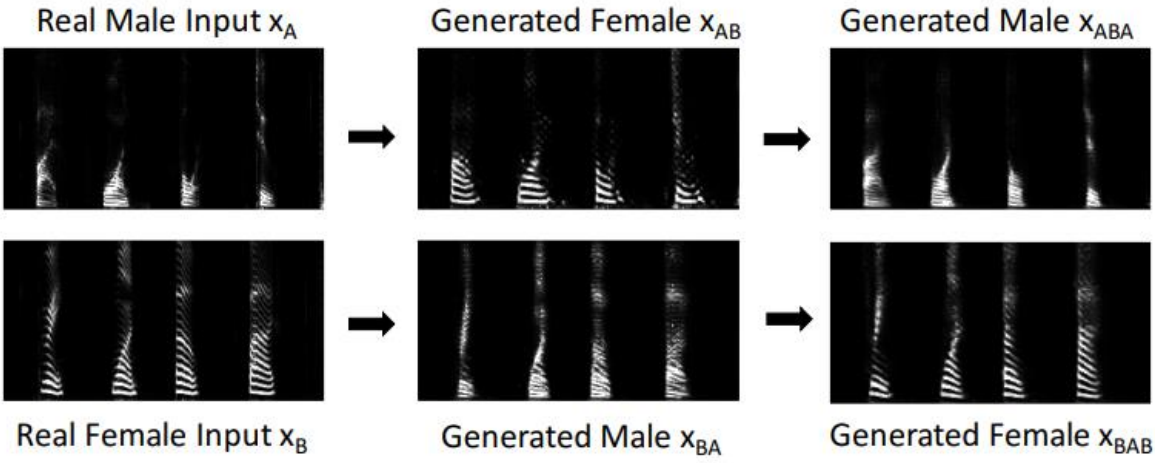


Рисунок 8 – Пример генерации гендерных особенностей голоса VoiceGAN

Это исследование открывает дорогу к новым достижениям в области имитации и изменения голосов, хотя и подчеркивает сложности в измерении и управлении такими преобразованиями.

3D моделирование с помощью генеративно состязательных сетей.

В статье "Изучение вероятностного латентного пространства форм объектов через 3D генеративно-состязательное моделирование" [20] представлен новый способ использовать генеративно состязательные сети. Они используются для создания трехмерных объектов, метод сочетает в себе глубокое обучение и сверточные сети для генерации реалистичных 3D объектов.

В статье были продемонстрированы три направления работы созданной сети:

– Первое – генерация 3d объектов по заданному классу, примеры генерации изображены на Рисунок 9.

– Второе – при обучении генератора дискриминатор научился определять класс 3д объекта, поэтому сеть можно использовать как классификатор объемных предметов.

– Третье – авторы статьи демонстрируют как сеть может реконструировать предметы по фотографии, пример работы такой сети изображен на Рисунок 10.

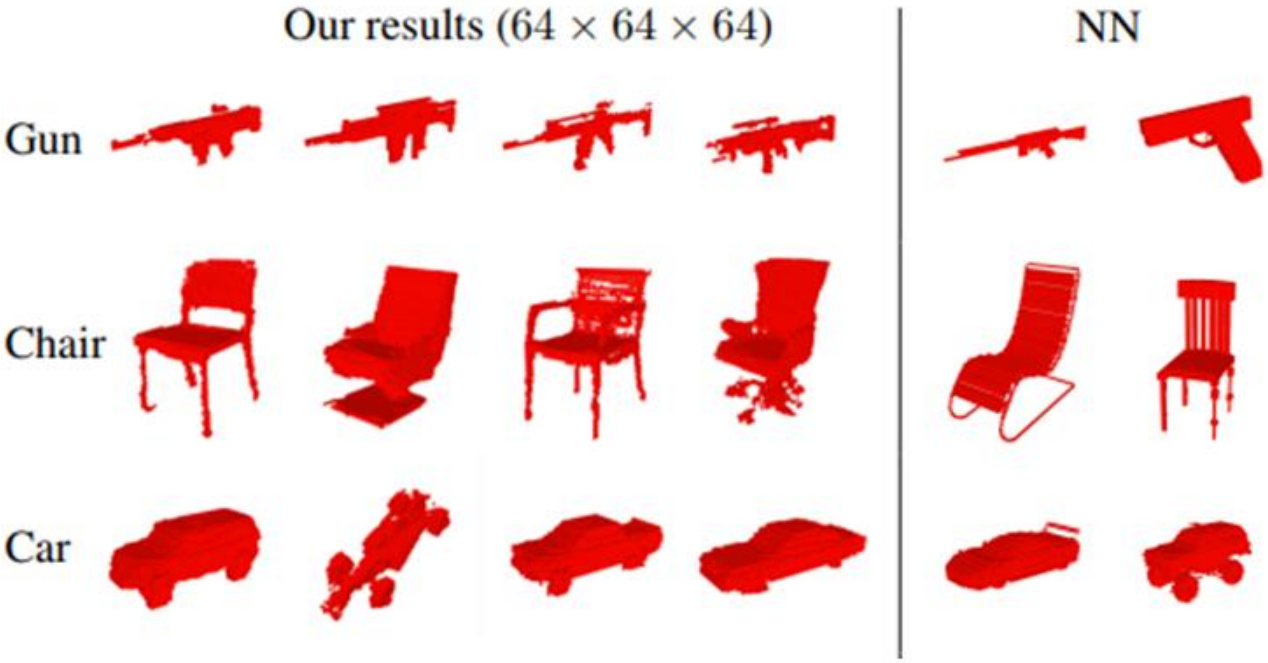


Рисунок 9 – Примеры генерации 3d объектов 3D GAN

Рисунок 10 – Примеры 3D реконструкции объемных предметов по фотографии

Исследование подчеркивает, что Генеративно-Состязательные Сети являются мощным инструментом в области 3D моделирования и могут использоваться в широком спектре приложений. Они демонстрируют впечатляющие результаты в генерации 3D объектов и распознавании форм, что делает их полезными во многих областях, от компьютерного зрения до автоматического проектирования.

Применение Генеративно-Состязательных Сетей в генерации уровней для видеоигр. Генеративно-Состязательные Сети так же применялись для генерации игровых уровней [1]. Этот подход предлагает новаторскую методику создания игрового контента, где искусственный интеллект не только имитирует, но и расширяет традиционные подходы к дизайну игр, предлагая уровни, которые могут быть одновременно разнообразными и привлекательными для игроков.

Исследование включало применение различных архитектур нейронных сетей. Сначала использовались свёрточные нейронные сети (CNN) для оценки сложности существующих уровней игры, затем – генеративные состязательные сети для создания новых уровней. Несмотря на первоначальные трудности с ограниченным объемом данных, методы, такие как DRAGAN [21], в конечном итоге показали успех в генерации базовых структур уровней. Также были использованы рекуррентные нейронные сети (RNN) и сети с долговременной памятью (LSTM) [22] для генерации уровней с контекстуальным пониманием.

В итоге, применение этих методов нейронных сетей привело к успешной генерации игровых уровней, которые были сложными и напоминали те, что разрабатывались дизайнерами. Примеры сгенерированных уровней изображены на Рисунок 11.

Рисунок 11 – Пример сгенерированного уровня GAN

Это исследование демонстрирует значительные возможности GAN в творческих процессах, таких как дизайн игровых уровней, и представляет собой важный шаг в интеграции GAN в разработку игр.

2 ТЕОРЕТИЧЕСКОЕ РЕШЕНИЕ

2.1 Требования к инструментарию и схема метода

Для проверки гипотезы и для дальнейшего исследования требовалось разработать инструментарий, который позволил бы проводить работу над инструментом генерации партиклей. Пайплайн должен включать не только саму генерацию эффектов, но и подготовку датасетов для обучения нейронной сети, возможность использовать генерацию нейронной сети в игровых движках.

Пайплайн состоит из следующих этапов:

– Подготовка датасетов для обучения нейронной сети,

– Экспорт датасетов и обучение нейронной сети,

– Использование результатов работы нейронной сети в игровом движке.

На Рисунок 12 Представлена высокоуровневая схема такого пайплайна.

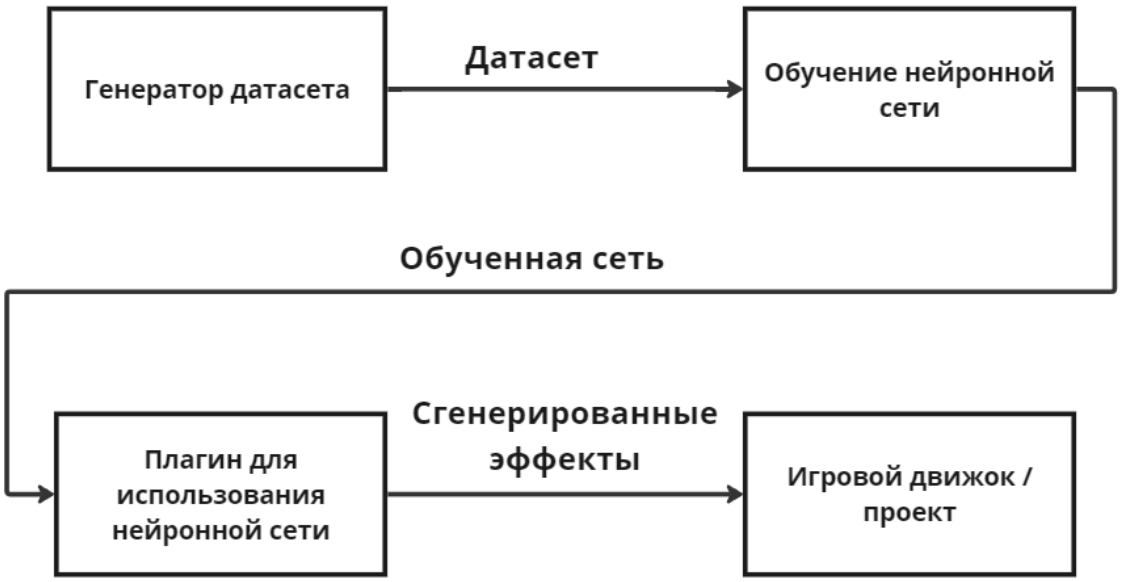


Рисунок 12 – Высокоуровневый пайплайн работы инструмента для генерации эффектов – систем частиц для игровых движков

На представленной схеме изображён процесс работы инструмента для генерации визуальных эффектов, предназначенного для использования в игровых движках. Процесс состоит из следующих этапов:

Генератор датасета – это инструмент, разработанный плагин для игрового движка Unity, который позволяет автоматически создавать обучающий датасет с учётом заданных пользователем параметров. На выходе из этого этапа мы получаем подготовленный датасет для обучения нейронной сети.

Обучение нейронной сети — это процесс, в котором используется полученный датасет для обучения нейронной сети. На этом этапе нейронная сеть учится генерировать новые эффекты, приближенные к эффектам из датасета. На выходе этого этапа мы получаем обученную нейронную сеть способную создавать эффекты.

Плагин для генерации для игровых движков – прежде чем использовать результаты генерации нейронной сети, требуется их преобразовать в вид используемый для партиклей в ассет игрового движка. И уже преобразованные сгенерированные эффекты можно использовать в проекте.

На Рисунок 13 Представлена углубленная схема такого пайплайна.

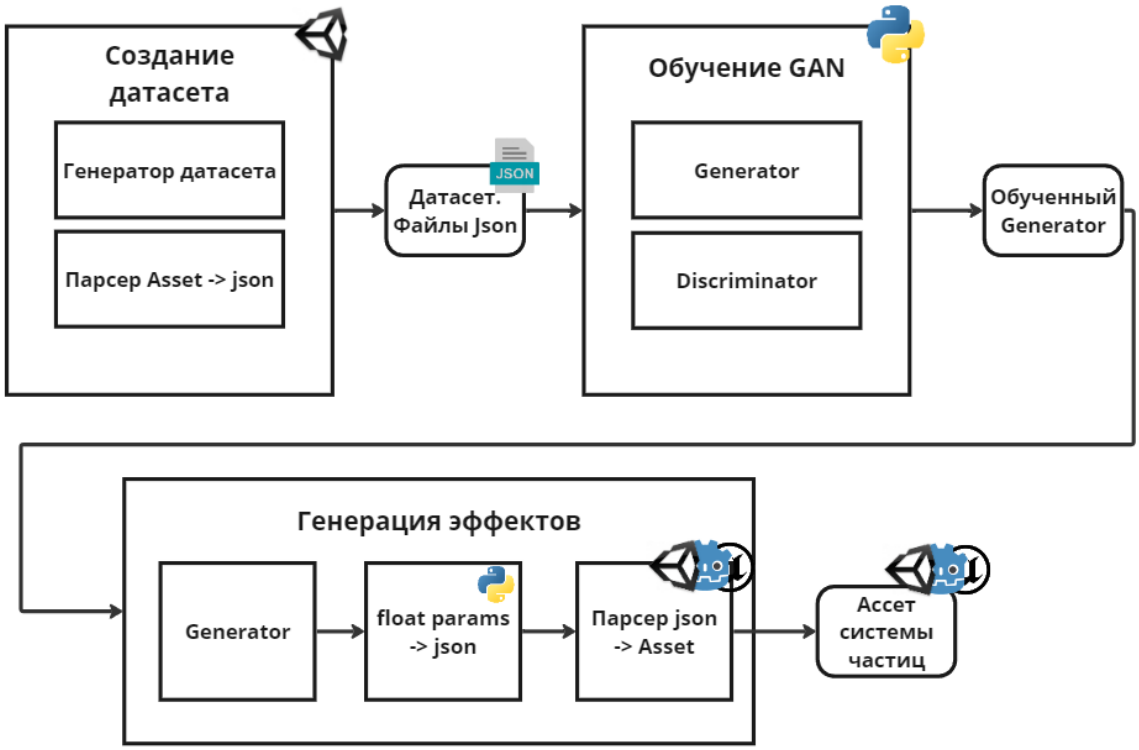


Рисунок 13 – Пайплайн работы инструмента для генерации ассетов систем частиц для игровых движков

Схема отражает инновационный подход к созданию визуальных эффектов, где искусственный интеллект нейронная сеть играет ключевую роль в автоматизации и улучшении процесса разработки VFX.

2.2 Требования представления систем частиц для обучения нейронной сети

Технические особенности работы нейронной сети ставят ограничение на размерность входных и выходных данных, но данные о системах частиц зачастую имеют разнообразное динамическое количество параметров, которое зависит от сложности воспроизводимого эффекта. Так же параметры отличаются в зависимости от игрового движка.

Соответственно для решения задачи, требуются инструменты позволяющие преобразовать параметры систем частиц в единый вид с установленным количеством и значением параметров. В рамках работы для единого стандарта были выбраны следующие модули систем частиц, которые

часто используются для моделирования партиклей в игровых движках и различных фреймворках для реализации партиклей [23] [24]:

– Время жизни частицы,

– Начальная скорость,

– Начальный размер,

– Начальное вращение,

– Начальный цвет,

– Эмиссия,

– Форма эммитера,

– Изменение скорости во времени,

– Изменение цвета во времени,

– Изменение размера во времени,

– Изменение вращения во времени.

Модули могут разбиваться на группы параметров, задаваться числовыми значениями, векторами или кривыми. На различных платформах и игровых движках, наборы могут отличаться, но эти модули в той или иной степени присутствуют и могут быть интерпретированы и интерполированы в зависимости от реализации соответствующего плагина для конкретного движка и присутствующих в движке параметров. Пример json с параметрами системы частиц изображен на Рисунок 14.

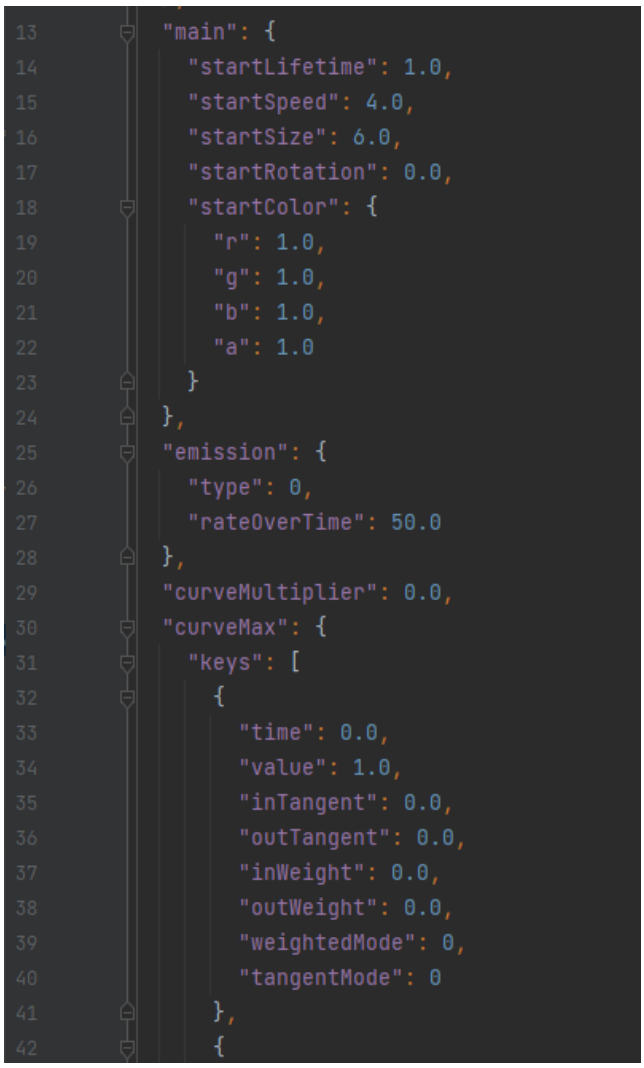


Рисунок 14 – Пример json с параметрами системы частиц

2.3 Схема работы модуля обучения нейронной сети и генерации

Один из этапов пайплайна это – обучение GAN: обучение генератора и дискриминатора с использованием сгенерированного датасета. Результатом работы этого этапа является обученный generator, который генерирует набор числовых параметров, которые описывают систему частиц. Схема работы модуля обучения изображена на Рисунок 15.

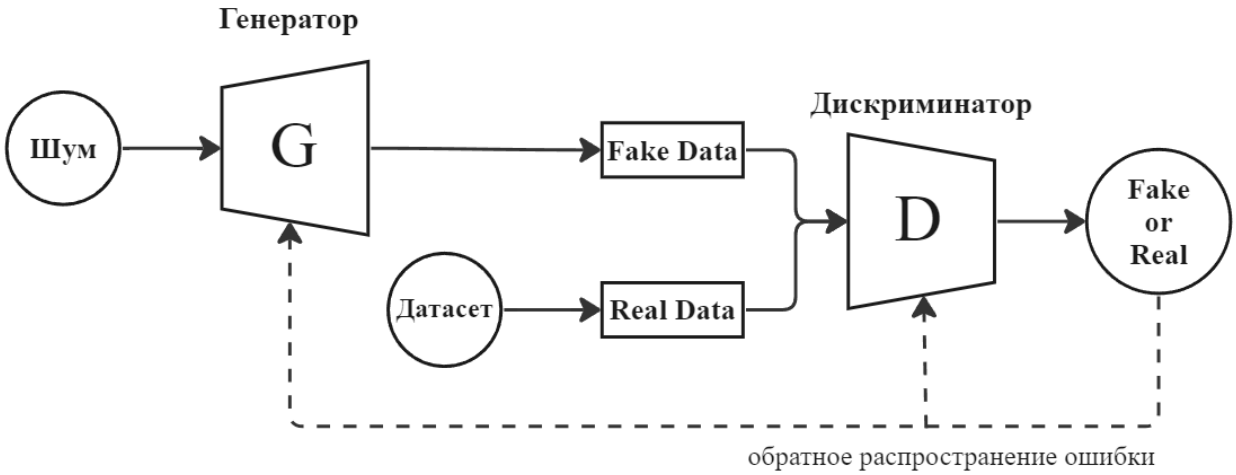


Рисунок 15 – Схема обучения GAN

Генератору на вход подается шумовой вектор, из которого генерируются синтетические данные. Так же берутся данные из датасета, проводится их нормализация для улучшения работы нейронной сети и передаются дискриминатору. Тот пытается спрогнозировать что было сгенерировано, и считается функция ошибки, и с помощью методов обратного распространения ошибки пересчитываются веса в генераторе и дискриминаторе.

Также в данной работе рассматривается возможность генерации визуальных эффектов по запросу, с указанием того, что ожидается в качестве результата от генератора. Дополнительная информация, которая помимо шума подаётся на вход генератору – теги.

Теги – это информация о системе частиц, которая описывает тот результат, который мы хотим получить, например если теги из датасета описывают цвет систем частиц или их форму, например зелёный взрыв или синий водопад, то генератор будет обучаться находить связи в этой информации и генерировать эффекты по таким запросам. Такая архитектура генеративно-состязательной нейронной сети называется conditional GAN и изображена на Рисунок 16.

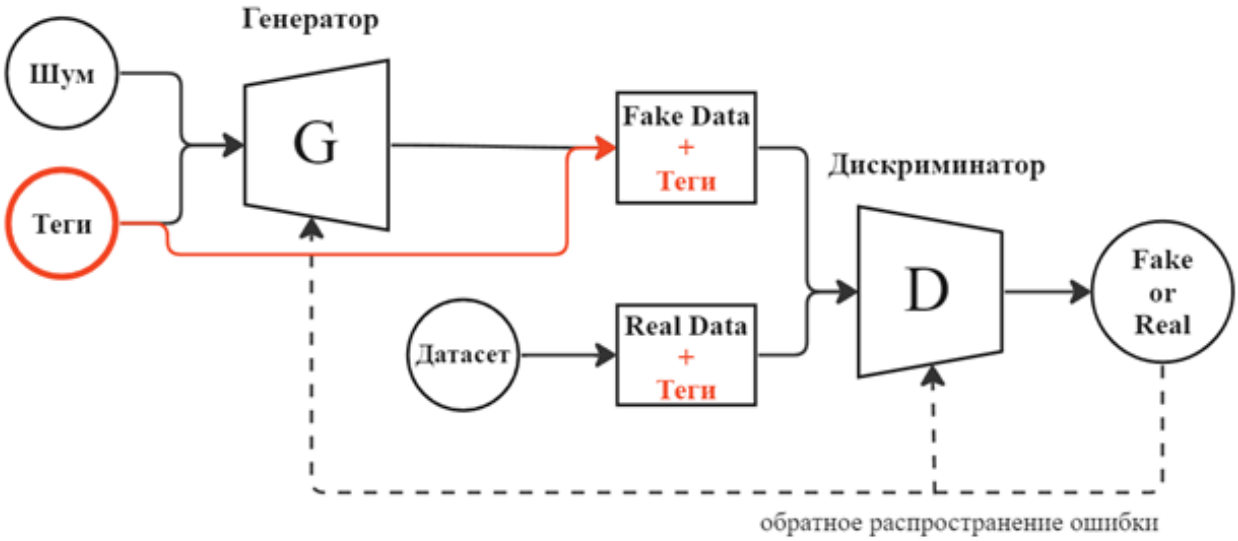


Рисунок 16 – Схема обучения conditional GAN

По итогам обучения получается обученный генератор, который может по запросу и условиям генерировать эффекты, которые после преобразования в ассеты игрового движка можно использовать в проекте. Процесс генерации эффектов изображен на Рисунок 17.

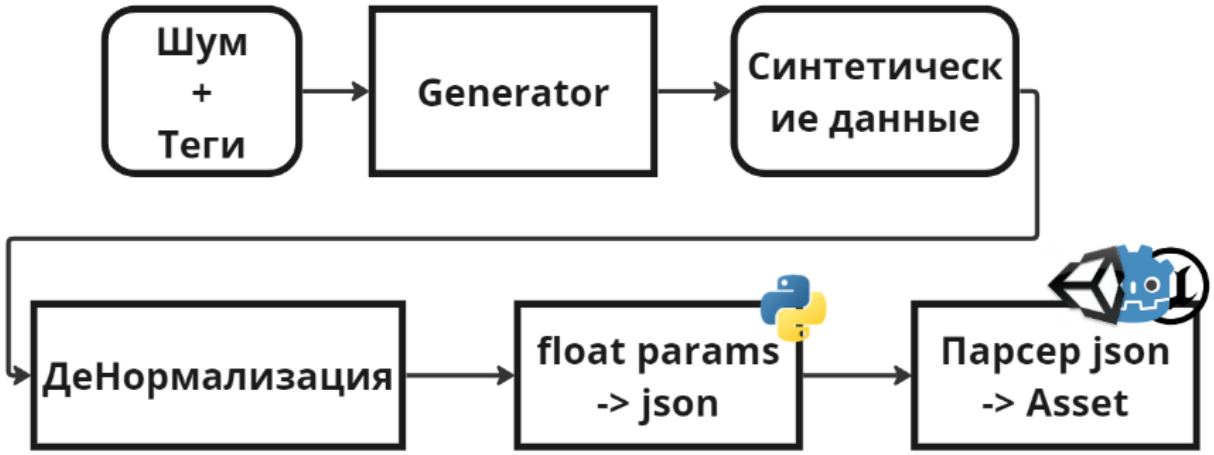


Рисунок 17 – Пайплайн генерации эффектов

3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНСТРУМЕНТА

3.1 Разработка инструмента для генерации датасетов партиклей для дальнейшего обучения

3.1.1 Требования к генератору датасетов

В рамках данного этапа, был разработан инструмент, результатом работы, которого будет датасет для обучения нейронной сети, который состоит из наборов числовых параметров, которыми описываются системы частиц в игровых движках, а также дополнительной информации, тэгах, по которым может производиться генерация. Были выдвинуты следующие требования для инструмента:

– Возможность использования разных наборов систем частиц,

– Приведение систем частиц к одному формату,

– Гибкая настройка формата частиц для удобства быстрых итераций при разработки нейронной сети,

– Гибкая настройка дополнительной информации – тегов для систем частиц.

Инструмент был разработан на базе игрового движка Unity. Он представляет из себя несколько модулей, которые управляют всем процессом создания датасета.

3.1.2 Конфигурационный файл датасета

Конфигурационный файл датасета – это объект, в котором хранятся системы частиц, ассеты движка на основе которых будет формироваться датасет. Интерфейс конфига датасета изображен на Рисунок 18.

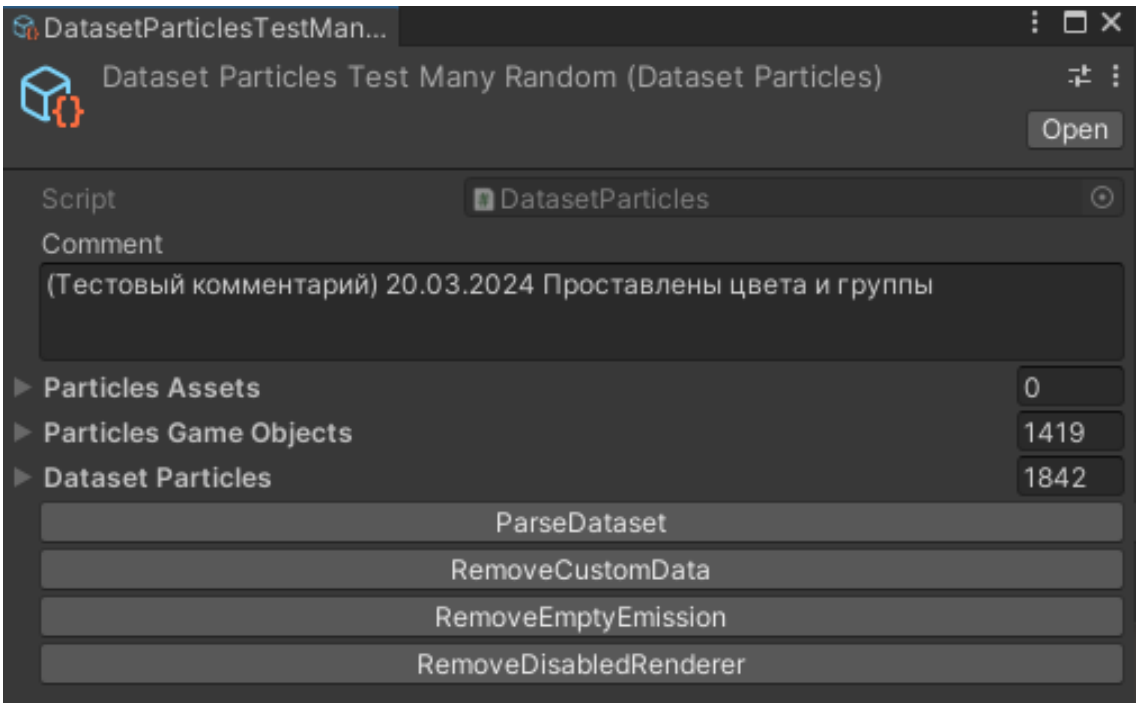


Рисунок 18 – Интерфейс конфига датасета со списком партиклов

Инструмент включает в себя следующие параметры:

– Комментарий для описания датасета. Вспомогательное поле, которое позволяет оставить дополнительную информацию для пользователя о том что находится в конфиге и на каком статусе готовности он находится.

– Particles Assets, Particles Game Objects – списки объектов в которые помещаются ассеты движка из которых будут выбираться системы частиц.

– Dataset particles – непосредственный список систем частиц который был выбран из указанных ассетов. Помимо самой системы частиц в каждом элементе списка хранится набор тэгов которые описывают данную систему частиц. Пример строчки датасета изображён на Рисунок 19.

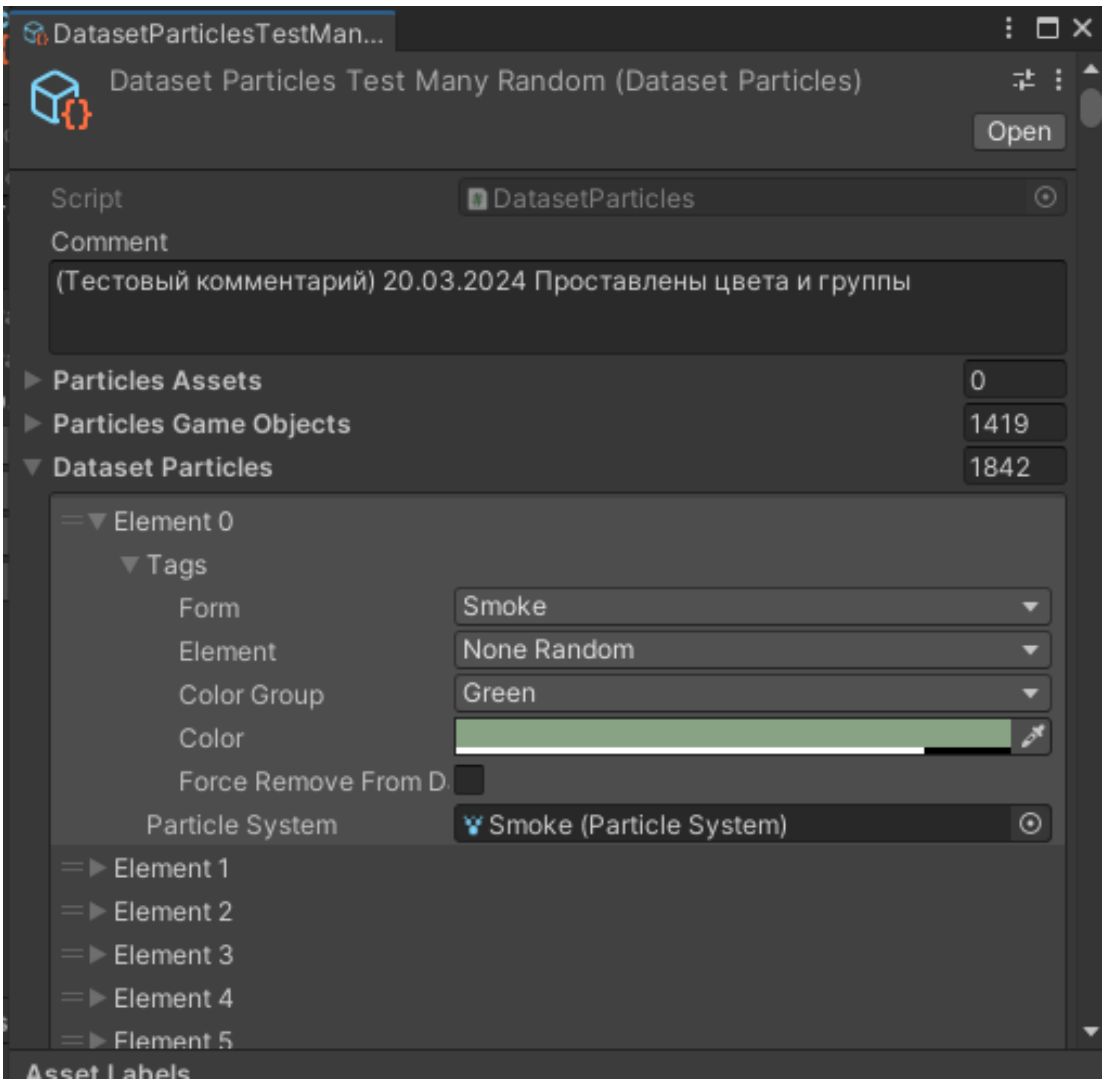


Рисунок 19 – Интерфейс конфига датасета с примером одной строки данных

Так же для конфига датасета были написаны дополнительные фильтры, позволяющие убрать из него нерелевантные системы частиц:

– RemoveCustomData – по этой кнопке из датасета убираются системы частиц у которых включен блок CustomData которые передаёт данные в шейдер, данные в этом блоке динамические и поскольку очень сложно разработать нейронную сеть способную обрабатывать динамическую длину данных такие системы частиц можно исключить из датасета, так как они могут негативно повлиять на обучение.

– RemoveEmptyEmission, RemoveDisabledRenderer – так как в используемых ассетах могут попадаться технические системы частиц,

которые не имеют визуала, и не имеют блок который отвечает за эмиссию частиц или отрисовку.

3.1.3 Модуль парсера ассетов партиклей в json

Парсер систем частиц – инструмент, который позволяет ассеты из формата движка преобразовывать в формат Json для использования в датасете, с настройкой преобразования параметров системы частиц. Интерфейс для тестирования модуля представлен на Рисунок 20.

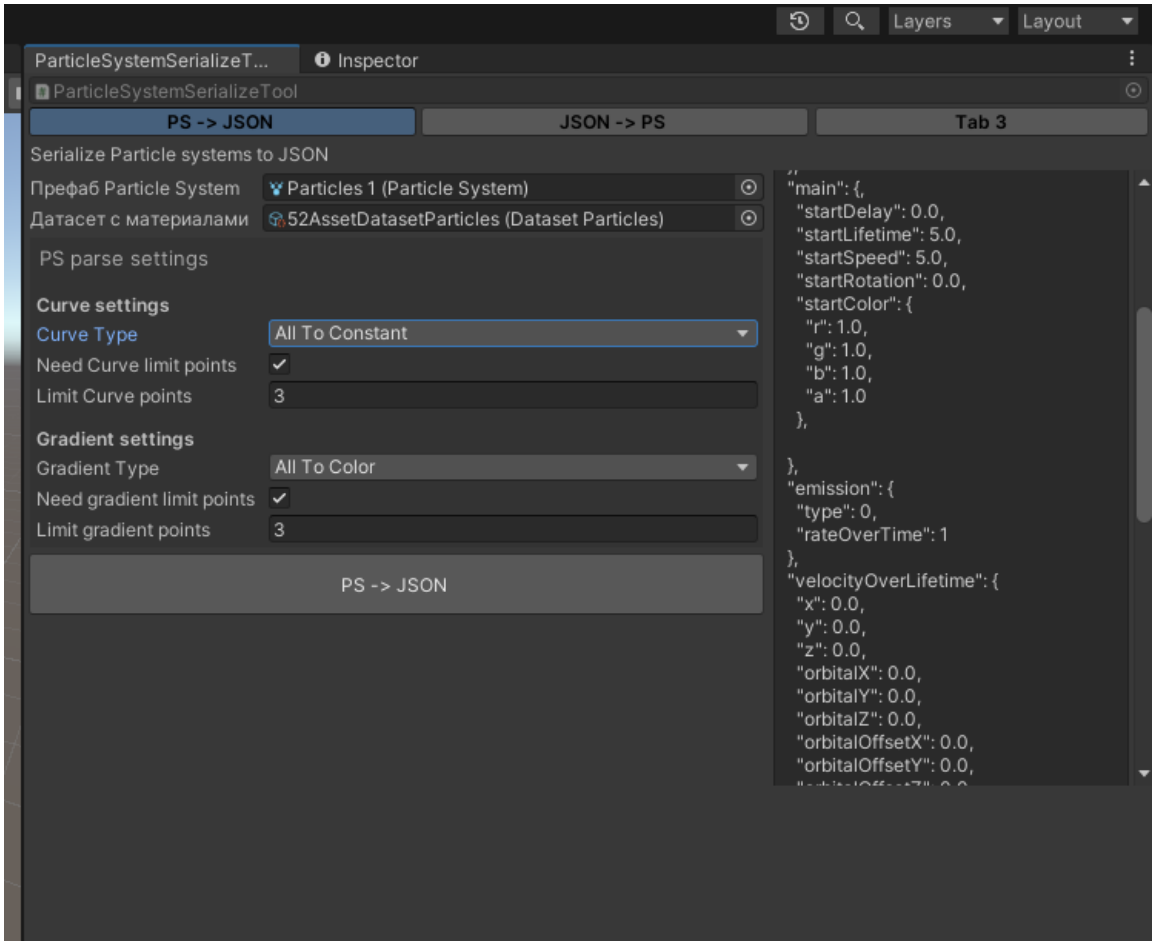


Рисунок 20 – Интерфейс инструмента парсера ассетов партиклей в json на игровом движке Unity

Список полей инструмента включает:

– Particle system – сам ассет параметры которого нужно преобразовать в json.

– Поле с специальным конфигурационным файлом Dataset Particles, в котором вручную указан список систем частиц из датасета. Содержание конфигурационного файла определяет список материалов, которые используются всеми партиклами в датасете. Материалы нумеруются для представления в числовом формате.

– PS parse settings – это блок параметров которые используются при обработки систем частиц для сохранения на диск. Т.к. для работы нейронной сети требуются унифицированные данные, все системы частиц должны иметь одинаковые размеры наборов параметров. Для этого в генераторе датасетов существуют настройки позволяющие менять итоговый вид, в который будут преобразованы некоторые из параметров системы частиц.

– Curve settings – это блок для настройки сохранения параметров имеющих тип MinMaxCurve. Который, может быть, нескольких видов – константа, случайное между двумя константами, одна кривая или случайное между двумя кривыми. В этом блоке можно выбрать в какой тип параметры такого типа во всех системах частиц преобразуются. А также указывается в какое количество точек будут преобразованы все кривые. Например, если выбрать тип AllToTwoCurves и ограничить кривые 2 точками, то все параметры интерполируются в две кривых каждая из которых будет иметь по 2 точки. Если значение параметра было константой, то после преобразования оно станет двумя одинаковыми кривыми с одинаковым значением. В таком подходе мы можем потерять некоторое количество информации о изначальных эффектах, но это требуется для приведения всех систем частиц к одному виду.

– Gradient settings – блок схожий по предназначению с Curve settings, но работает с параметрами описывающими цвет. У градиентов есть следующие виды: один цвет, случайный между двумя цветами и градиент с несколькими точками изменения цвета. Мы так же можем выбрать в каком формате будут сохранены такие поля и ограничить кол-во точек в градиентах при необходимости.

3.1.4 Модуль выбора датасета и генерации json

Второй модуль отвечает за выбор нужных ассетов, преобразование их в json и сохранения датасета на диск. Интерфейс инструмента для генерации датасетов изображен на Рисунок 21.

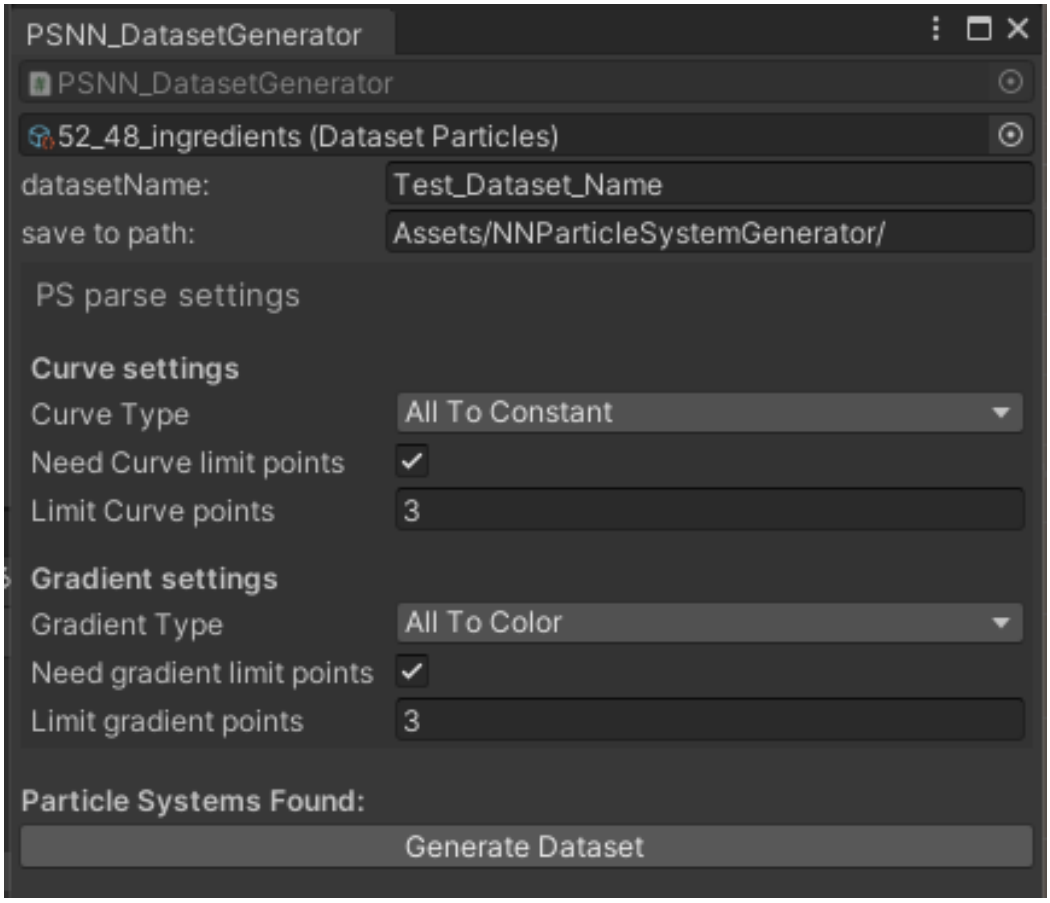


Рисунок 21 – Интерфейс инструмента для генерации датасетов на игровом движке Unity

Инструмент включает в себя следующие параметры:

– Используемые системы частиц – указывается специальный конфигурационный файл Dataset Particles, в котором вручную указаны нужные системы частиц.

– PS parse settings – это блок параметров которые используются при обработки систем частиц для сохранения на диск.

– Dataset Name – поле в которому указывается название датасета, которое будет использоваться для сохранения файлов.

– Save to path – указывается путь, куда нужно сохранить результат работы генератора датасета.

– Кнопка Generate Dataset – по которой начинается генерация датасета – приведения параметров систем частиц к одному виду и сохранению результатов в формате json на диск.

3.1.5 Инструмент расстановки тэгов для датасета

Для обучения нейронной сети генерации эффектов по указанным тэгам, нужно подготовить датасет примеров эффектов с расставленными тэгами. Поскольку в рамках данной работы нужно создать датасет, был необходим удобный инструмент расстановки тэгов, который позволит быстро и продуктивно производить расстановку тэгов. Такой инструмент был разработан и изображён на Рисунок 22.

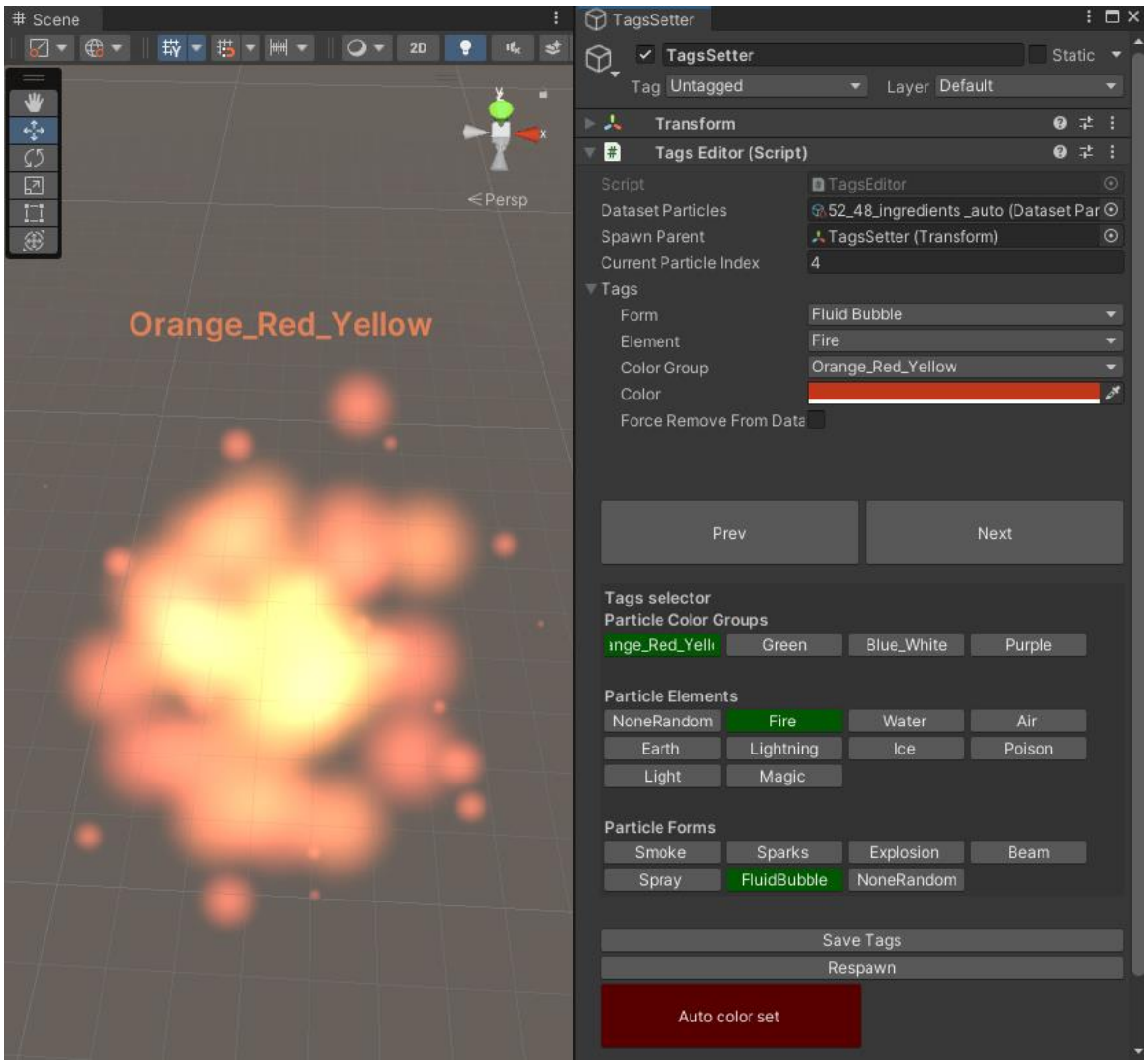


Рисунок 22 – Интерфейс инструмента расстановки тэгов для датасета

Интерфейс инструмента расстановки тэгов состоит из следующих элементов:

– DatasetParticlesConfig – поле в котором указывается конфиг с датасетом систем частиц для которых нужно расставить тэги.

– SpawnParent – объект на сцене внутрь которого будут помещаться системы частиц для визуализации пользователю.

– CurrentParticleIndex – индекс в списке систем частиц в датасете текущего эффекта на сцене. С помощью этого поля можно напрямую отобразить нужный эффект.

– Tags – поле с информацией и тэгами описывающими данный эффект.

– Кнопки Next Prev – кнопки, которые меняют текущий эффект на один следующий или предыдущий.

– Tags Selector – блок в котором располагаются кнопки для быстрого выбора тэга. Текущий выбранный тэг подсвечен зелёным.

– Save Tags – кнопка принудительного сохранения переноса в датасет указанной в Tags информации. Сохранение так же производится автоматически при изменении данных и нажатию кнопок Next Prev.

– Respawn – кнопка по которой текущий эффект удаляется и появляется заново. Можно использовать чтобы перезапустить проигрывание эффекта или чтобы восстановить изменённый на сцене.

– Auto color set – вспомогательный инструмент который автоматически пытается определить доминантный цвет системы частиц и группу эффекта по цвету. Результат работы, построенный на простейшем алгоритме, часто даёт сбои, но может ускорить расстановку цветовой группы.

Принцип работы расстановки тэгов заключается в:

– С помощью кнопок Prev и Next переключение текущего эффекта, поля с информацией о эффекте и кнопки тэгов заменяются на информацию нового выбранного эффекта.

– Эффект помещается на сцену и проигрывается для расставляющего. Предыдущий эффект удаляется со сцены.

Расставляющий пользователь выбирает нужные тэги и информацию о системе частиц и сохраняет результат.

3.2 Инструмент генерации для игрового движка

Этап генерации эффектов предназначен для использования результатов работы генератора – преобразование их в ассеты выбранного движка. В рамках этого этапа был разработан инструмент на Python который результат работы генератора представляет в исходном json из датасета, а также парсер на игровом движке Unity который преобразовывает json в ассет системы частиц для использования в движке.

Технически результатом работы генератора является массив чисел, которые являются параметрами полей, которые описывает систему частиц. Для дальнейшего использования сгенерированных чисел, нужно собрать из них исходный json, на котором обучалась нейронная сеть. Для этого был написан скрипт на python который собирает json по примеру из датасета. Пример работы модуля, по заполнению json числами изображен на Рисунок 23.

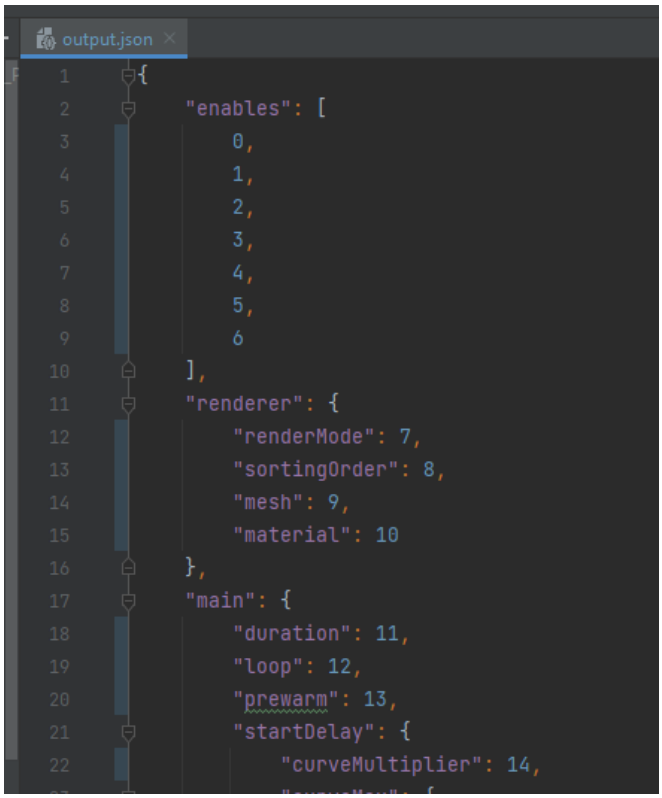


Рисунок 23 – Пример заполнения json числами

Так же для тестирования и исследования было необходимо реализовать инструмент – парсер позволяющий быстро работать с параметрами систем частиц, проверять результаты генерации нейронной сети, а также для генерации с использованием тэгов. Инструмент был реализован на движке Unity. Интерфейс инструмента изображён на Рисунок 24.

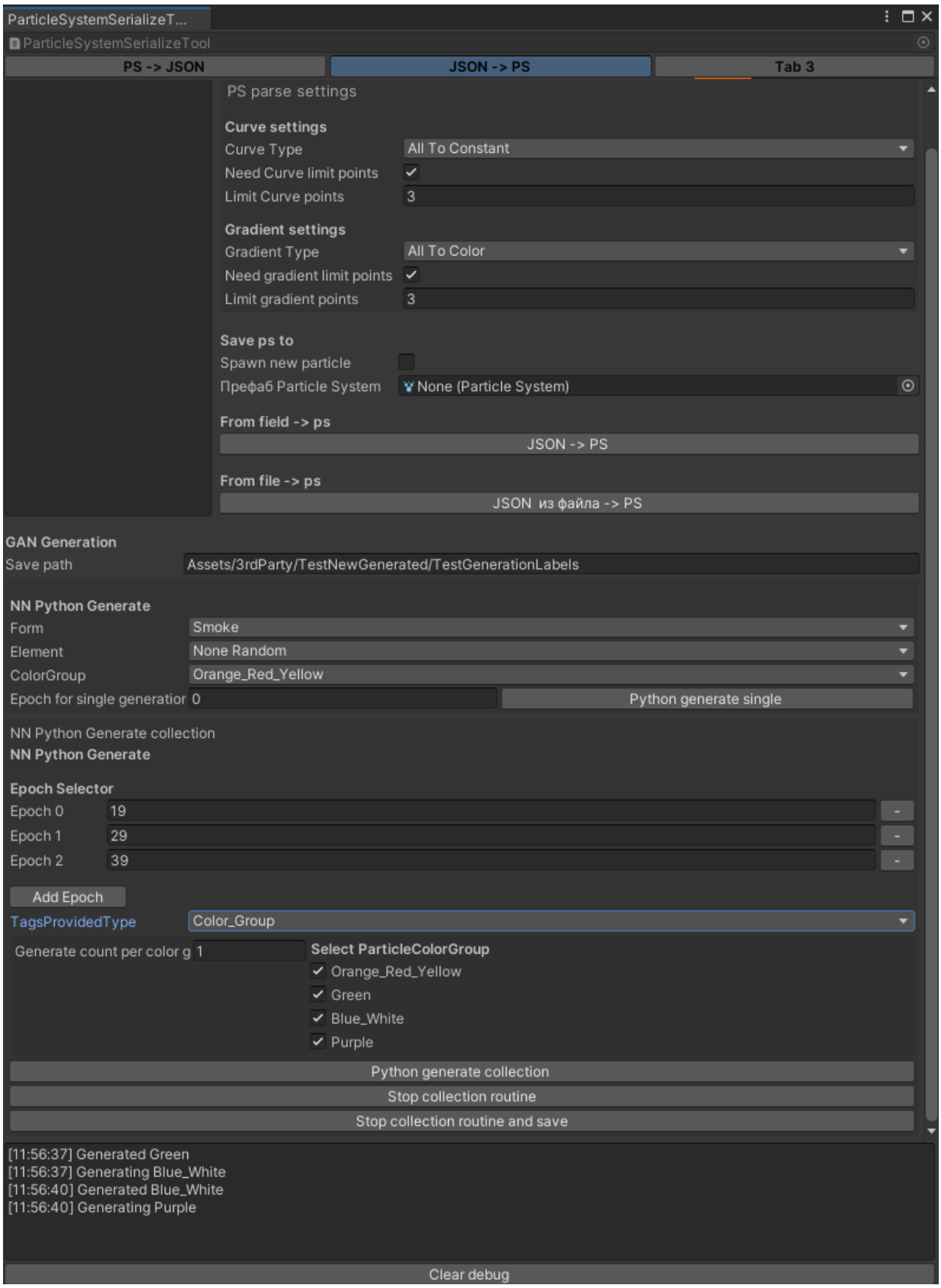


Рисунок 24 – Инструмент для работы с генерацией

Реализованный инструмент позволяет, используя указанный датасет, преобразовать JSON параметров системы частиц в ассет для игрового движка Unity.

Реализовано три режима работы:

– Преобразовать указанный в поле json в ассет партиклей,

– Преобразовать json из указанного файла в ассет партиклей,

– Запустить в работу нейронную сеть генератор и сгенерированный результат преобразовать в ассет партиклей.

Инструмент содержит следующие поля:

– Поле для ручного ввода Json,

– Поле для указания датасета из которого будут браться материалы для генерации системы частиц,

– Параметры для парсинга json,

– Блок сохранения ассета. Инструмент предоставляет две опции, первая создает новый ассет с системой частиц, а вторая заполняет данными из генерации уже существующий ассет.

– Кнопка преобразования json из текстового поля,

– Парсинг из указанного файла с json,

– GAN Generation. Блок для работы с нейронной сетью, состоит из двух блоков: Одиночной генерации эффекта или генерации коллекции эффектов.

– Блок для указания эпохи нейронной сети, которую нужно использовать для генерации.

– Одиночная генерация состоит из полей тэгов, на основе которых нужно сгенерировать новый эффект. А также поле с номером эпохи и кнопки запуска генерации.

– Генерация коллекции эффектов. Эта секция позволяет сгенерировать сразу несколько эффектов по указанным параметрам. Указывается список эпох, выбираются нужные тэги и количество эффектов для каждого тэга. Эта секция может значительно ускорить итерации разработки нейронной сети и

выбор разных эффектов. Так же имеются кнопки управлением процессом генерации, возможность остановить генерацию, а также консоль с выводом сообщений о состоянии генерации. Пример генерации коллекции изображён на Рисунок 25.

Рисунок 25 – Результат работы секции генерации коллекции эффектов по тэгам и нескольким эпохам

Таким образом был реализован инструмент на игровом движке Unity, который позволяет удобно работать с json представлениями систем частиц и быстро проверять работу нейронной сети, по кнопке генерируя новые эффекты.

4 ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ И ГЕНЕРАЦИЯ

4.1 Проверка гипотезы о возможности генерации систем частиц

Для проведения экспериментов был подготовлен датасет из бесплатных наборов VFX для игрового движка юнити. Использовался бесплатный ассет с эффектами [2], примеры эффектов изображены на Рисунок 26.

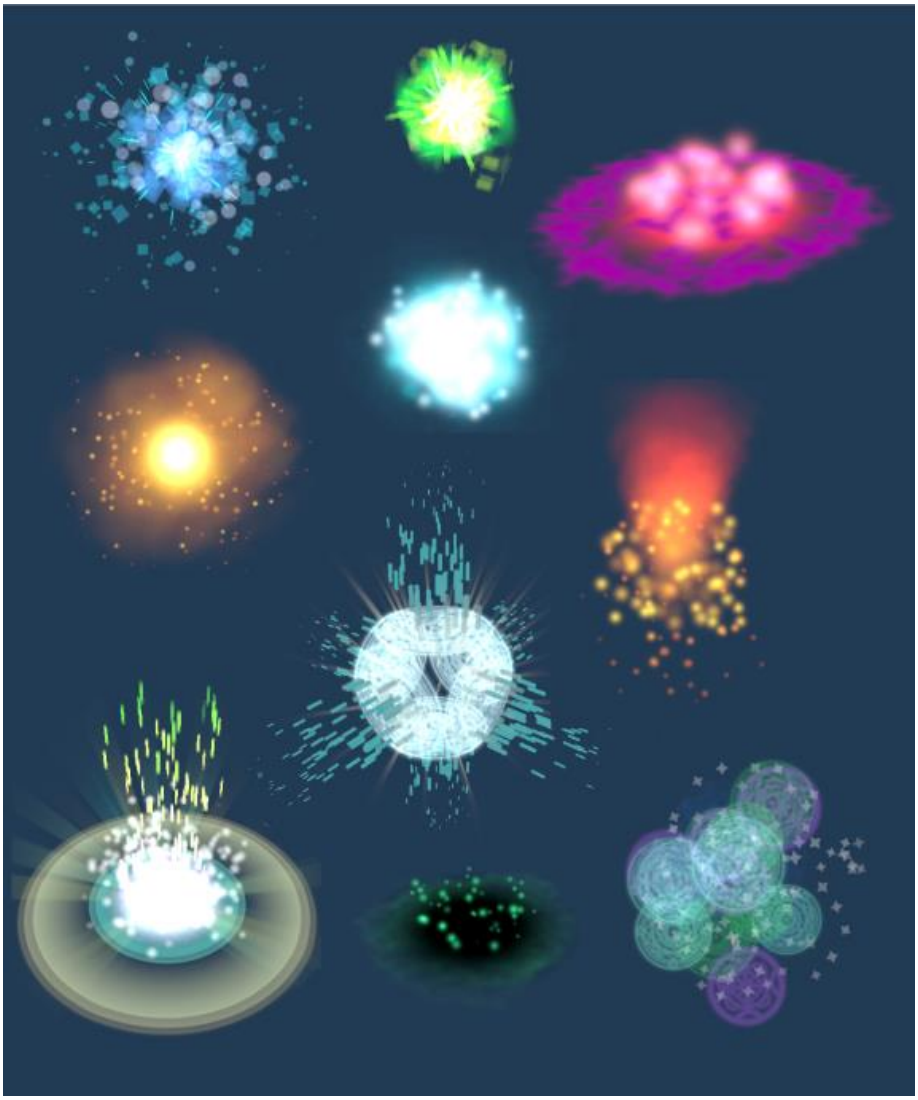


Рисунок 26 – Примеры комплексных эффектов из ассета

Эффекты из ассета являются достаточно комплексными, потому что они состоят из нескольких эффектов, в рамках данного исследования было допущено упрощение задачи и все комплексные эффекты были разобраны на

составляющие и использовались для генерации. Примеры атомарных эффектов, из которых собирался датасет изображены на Рисунок 27.

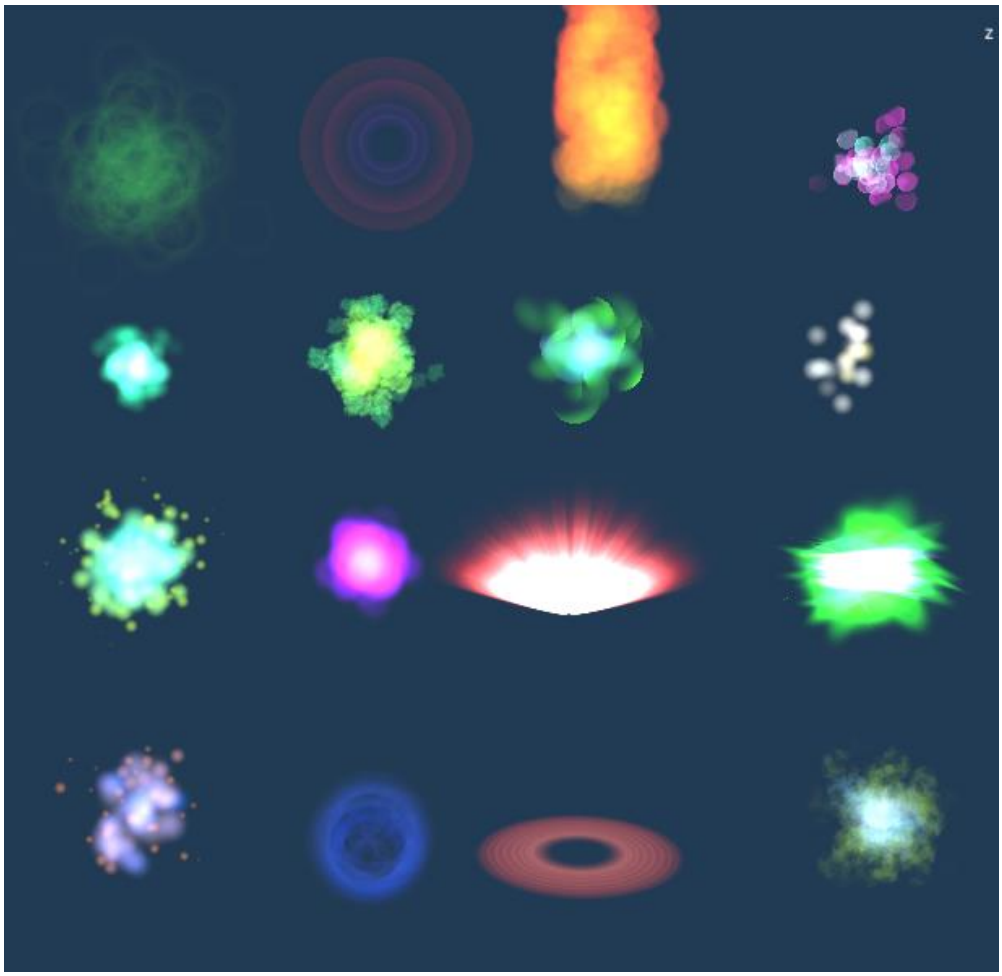


Рисунок 27 – Пример атомарных эффектов для датасета

Из атомарных эффектов был подготовлен датасет из 425 эффектов.

В ходе исследования было проведено множество итераций и работы над архитектурой нейронной сети и методов обучения. Для быстрой и удобной итерации разработки нейронной сети были выбраны гиперпараметры, с помощью которых можно влиять на архитектуру генератора и дискриминатора:

– Размер шумового вектора, подающегося на вход генератору

– Размер первого скрытого слоя генератора и дискриминатора

– Количество скрытых слоев генератора

– Количество скрытых слоев дискриминатора

Размеры скрытых слоев вычисляются по формуле:

$$S * 2^n$$

где n принимает значения от 0 до количества скрытых слоев, а S – размер первого скрытого слоя. Например, с S = 128 и n = 3 получатся слои с размерами: 128, 256, 512, 1024.

На данном этапе итогом итераций является архитектура генератора, которая представляет собой линейные слои с ReLU функциями активации, слоями нормализации и слоями Dropout в середине, и последний завершающий слой имеет функцию активации Tanh чтобы итоговые значения были в диапазоне от -1 до 1. Архитектура генератора изображена на Рисунок 28.

```
==========================================================================
Generator                                [8, 2197]                 --
├─Sequential: 1-1                        [8, 128]                  --
│    └─Linear: 2-1                       [8, 16]                   336
│    └─InstanceNorm1d: 2-2               [8, 16]                   --
│    └─ReLU: 2-3                         [8, 16]                   --
│    └─Dropout1d: 2-4                    [8, 16]                   --
│    └─Linear: 2-5                       [8, 32]                   544
│    └─InstanceNorm1d: 2-6               [8, 32]                   --
│    └─ReLU: 2-7                         [8, 32]                   --
│    └─Dropout1d: 2-8                    [8, 32]                   --
│    └─Linear: 2-9                       [8, 64]                   2,112
│    └─InstanceNorm1d: 2-10              [8, 64]                   --
│    └─ReLU: 2-11                        [8, 64]                   --
│    └─Dropout1d: 2-12                   [8, 64]                   --
│    └─Linear: 2-13                      [8, 128]                  8,320
│    └─InstanceNorm1d: 2-14              [8, 128]                  --
│    └─ReLU: 2-15                        [8, 128]                  --
│    └─Dropout1d: 2-16                   [8, 128]                  --
├─Sequential: 1-2                        [8, 2197]                 --
│    └─Linear: 2-17                      [8, 2197]                 283,413
│    └─Tanh: 2-18                        [8, 2197]                 --
==========================================================================
Total params: 294,725
```

Рисунок 28 – Архитектура Генератора

Архитектура дискриминатора также представляет собой линейные слои с ReLU функциями активации, и слоями нормализации, и заверщающие функцтей активации Sigmoid для определения прогноза. Архитектура дискриминатора изображена на Рисунок 29. Результаты генерации изображены на Рисунок 30.

```
==========================================================================
Discriminator                            [8, 1]                    --
├─Sequential: 1-1                        [8, 1]                    --
│    └─Linear: 2-1                       [8, 32]                   2,560
│    └─InstanceNorm1d: 2-2               [8, 32]                   --
│    └─ReLU: 2-3                         [8, 32]                   --
│    └─Linear: 2-4                       [8, 16]                   528
│    └─InstanceNorm1d: 2-5               [8, 16]                   --
│    └─ReLU: 2-6                         [8, 16]                   --
│    └─Linear: 2-7                       [8, 1]                    17
│    └─Sigmoid: 2-8                      [8, 1]                    --
==========================================================================
Total params: 3,105
```

Рисунок 29 – Архитектура Дискриминатора

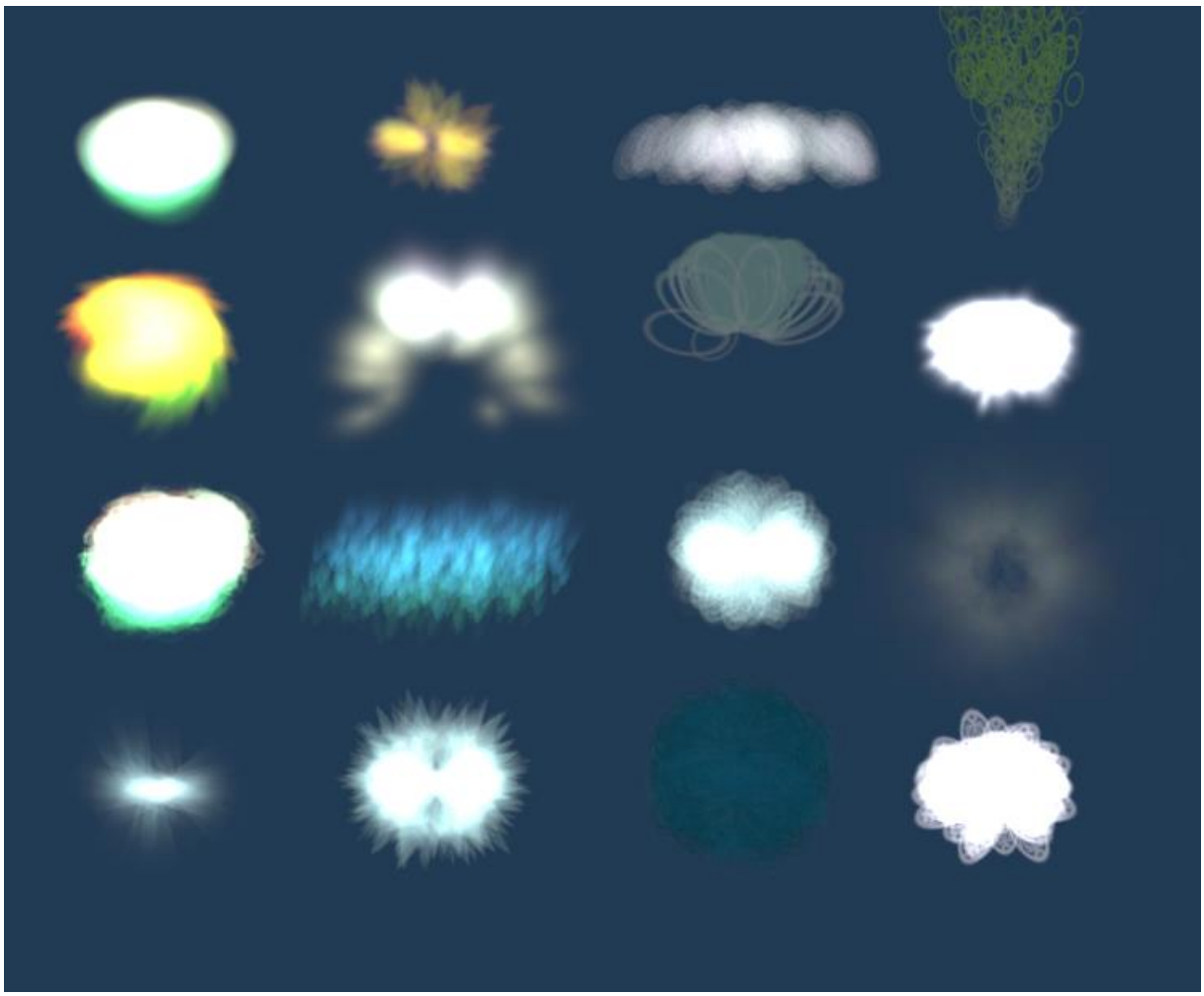


Рисунок 30 – Примеры сгенерированных эффектов

Способность нейронной сети автоматически создавать качественные VFX ассеты, позволяет значительно ускорить процесс разработки, снизив временные и финансовые затраты на разработку систем частиц. Этот проект демонстрирует перспективы использования нейронных сетей в сфере создания визуальных эффектов для игр и предоставляет новые возможности для индустрии игровой разработки.

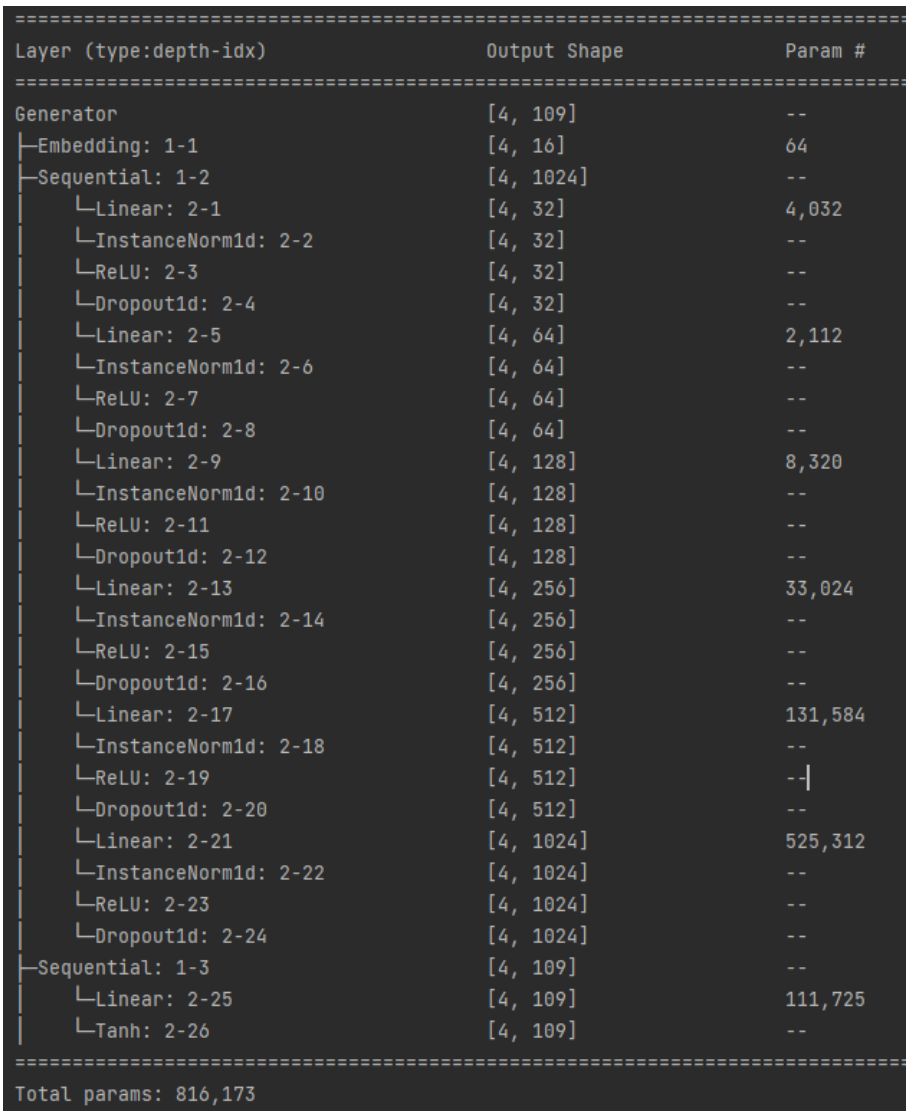
Дальнейшие перспективы и направления работы над исследованием могут заключаться в улучшении архитектуры нейронной сети, поиске более эффективных подходов к обучению генеративно состязательной сети, с целью увеличения разнообразия генераций и добавления дополнительного контроля генерации, путем использования условных генеративно состязательных сетей.

4.2 Генерация по тегам

Для обучения нейронной сети способной использовать указанные теги для генерации эффектов, необходимо реализовать условную генеративно-состязательную нейронную сеть (cGAN) [25] [26]. В модули генератора и дискриминатора необходимо добавить возможность принимать на вход информационные теги, которые будут описывать нужный эффект.

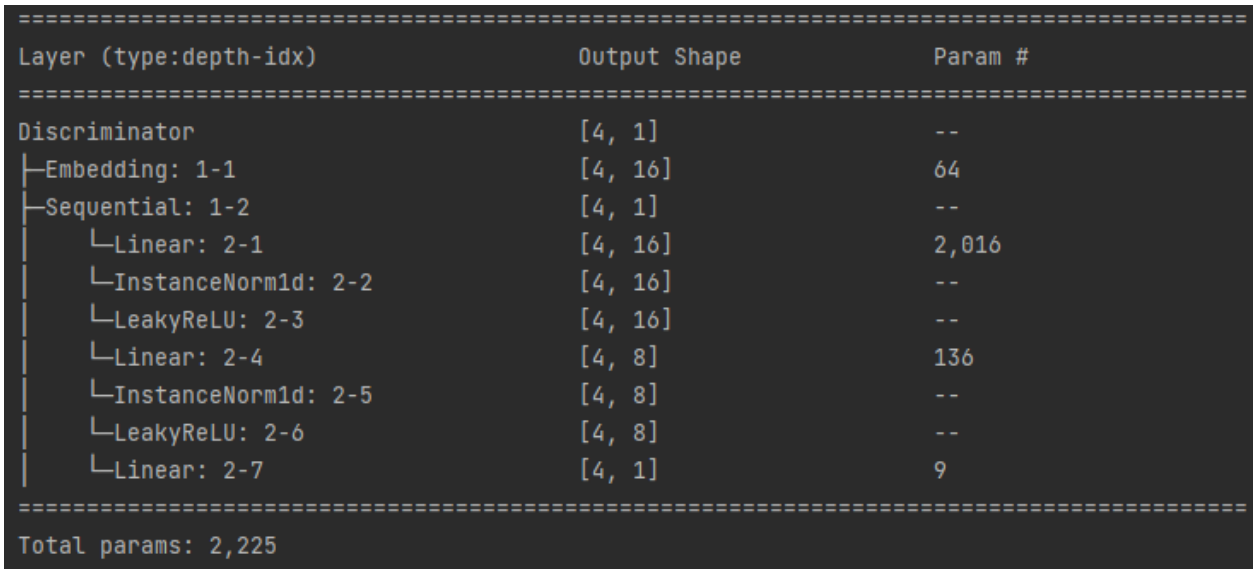
Для работы с тегами в сеть добавляются дополнительные слои эмбеддинга (Embeddings) [27], которые позволяют преобразовать категориальные данные в векторные представления, в нашем случае тэги описывающие эффекты. Векторные представления тэгов позволяют нейронной сети увидеть дополнительную информацию о связи этих меток и эффекта.

Новая архитектура слоёв генератора и дискриминатора изображена на Рисунок 31 и Рисунок 32.



```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
Generator                                [4, 109]                  --
├─Embedding: 1-1                         [4, 16]                   64
├─Sequential: 1-2                        [4, 1024]                 --
│    └─Linear: 2-1                       [4, 32]                   4,032
│    └─InstanceNorm1d: 2-2               [4, 32]                   --
│    └─ReLU: 2-3                         [4, 32]                   --
│    └─Dropout1d: 2-4                    [4, 32]                   --
│    └─Linear: 2-5                       [4, 64]                   2,112
│    └─InstanceNorm1d: 2-6               [4, 64]                   --
│    └─ReLU: 2-7                         [4, 64]                   --
│    └─Dropout1d: 2-8                    [4, 64]                   --
│    └─Linear: 2-9                       [4, 128]                  8,320
│    └─InstanceNorm1d: 2-10              [4, 128]                  --
│    └─ReLU: 2-11                        [4, 128]                  --
│    └─Dropout1d: 2-12                   [4, 128]                  --
│    └─Linear: 2-13                      [4, 256]                  33,024
│    └─InstanceNorm1d: 2-14              [4, 256]                  --
│    └─ReLU: 2-15                        [4, 256]                  --
│    └─Dropout1d: 2-16                   [4, 256]                  --
│    └─Linear: 2-17                      [4, 512]                  131,584
│    └─InstanceNorm1d: 2-18              [4, 512]                  --
│    └─ReLU: 2-19                        [4, 512]                  --
│    └─Dropout1d: 2-20                   [4, 512]                  --
│    └─Linear: 2-21                      [4, 1024]                 525,312
│    └─InstanceNorm1d: 2-22              [4, 1024]                 --
│    └─ReLU: 2-23                        [4, 1024]                 --
│    └─Dropout1d: 2-24                   [4, 1024]                 --
├─Sequential: 1-3                        [4, 109]                  --
│    └─Linear: 2-25                      [4, 109]                  111,725
│    └─Tanh: 2-26                        [4, 109]                  --
==========================================================================
Total params: 816,173
```

Рисунок 31 – Архитектура Генератора со слоями Embedding



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Discriminator                            [4, 1]                    --
├─Embedding: 1-1                         [4, 16]                   64
├─Sequential: 1-2                        [4, 1]                    --
│    └─Linear: 2-1                       [4, 16]                   2,016
│    └─InstanceNorm1d: 2-2               [4, 16]                   --
│    └─LeakyReLU: 2-3                    [4, 16]                   --
│    └─Linear: 2-4                       [4, 8]                    136
│    └─InstanceNorm1d: 2-5               [4, 8]                    --
│    └─LeakyReLU: 2-6                    [4, 8]                    --
│    └─Linear: 2-7                       [4, 1]                    9
==========================================================================================
Total params: 2,225
```

Рисунок 32 – Архитектура Дискриминатора со слоями Embedding

В рамках данного этапа производились итерации обучения нейронной сети, изменялись её гиперпараметры, чтобы добиться успешной генерации по тэгам. Генеративно-состязательная сеть с условием добавляет новые гиперпараметры, которые регулируют размер вектора embeddings на который раскладываются указанные тэги.

Для проверки гипотезы о возможности обучения нейронной сети поставленной задаче, условия генерации были упрощены. Нейронная сеть должна сгенерировать эффект из указанной группы. Эффекты были поделены на четыре группы по цветовому признаку.

В результате итераций на данном этапе сеть научилась создавать эффекты для каждой группы, которые обладали уникальными для группы свойствами, цветом частиц.

Обучение производилось на датасете размером 1053 записи с шагом обучения (learning rate) 5e-5, и упрощённым набором параметров – каждая запись имела 109 параметров. Были упрощены все кривые до одного среднего числа, а все цветовые градиенты до одного среднего цвета.

Датасет до упрощения изображён на Рисунок 33. Результаты генерации упрощённых эффектов по цветовым группам изображены на Рисунок 34.

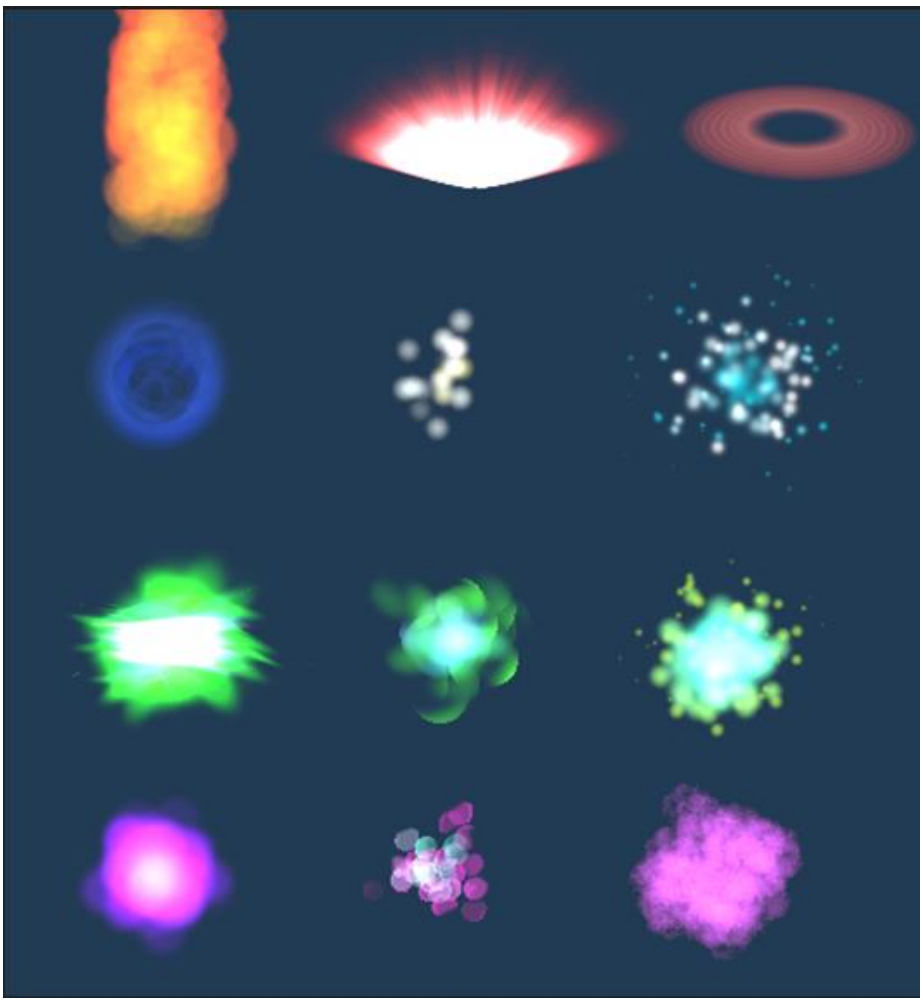


Рисунок 33 – Размеченный по цветовым группам датасет. Каждая строка – цветовая группа (Orange_Red_Yellow, Blue_White, Green, Purple)

Группы были выбраны на основе количества различных систем частиц подходящим под свойства, так как важно, чтобы количество элементов в каждой группе не было низким, так как это может повлиять на качество обучения, нейронной сети может не хватить данных, чтобы научиться находить связи между параметрами эффектов в одной группе.

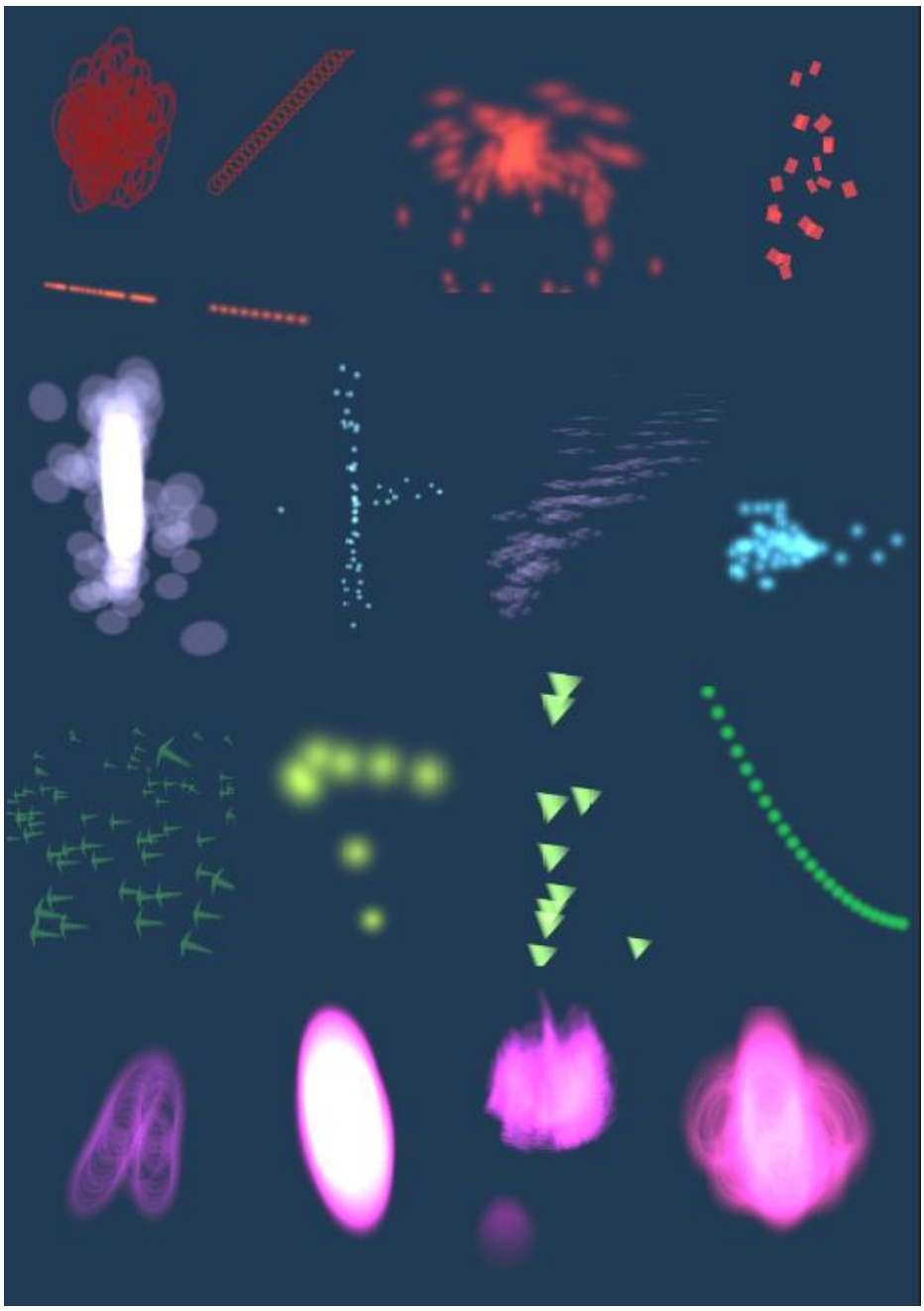


Рисунок 34 – Результаты генерации. Каждая строка – цветовая группа (Orange_Red_Yellow, Blue_White, Green, Purple)

4.3 Разработка метрик оценки генерации

Для оценки результатов генерации необходимо разработать программно-считаемую метрику, которая сможет описать качество и разнообразие результатов работы нейронной сети.

Технически результатом работы генератора является массив чисел, которые являются параметрами полей описывающих систему частиц.

Соответственно исходные данные для метрики — это массив наборов таких чисел, которые описывают сгенерированные эффекты. В рамках данного этапа были рассмотрены несколько подходов оценки. Но многие подходы имеют свои минусы и крайние точки, на которых могут нереалистично и некачественно производить оценку.

Разнообразие среднего по параметрам. Самая простая и быстрая в реализации метрика, которая использовалась в первых итерациях. Для оценки из датасета брался случайный набор эффектов, брался экземпляр нейронной сети и генерировалось столько же эффектов. Для каждого эффекта считалось среднее значение по всем параметрам, округлялось до 2 знаков после запятой и считалось количество уникальных таких средних. Итого получалась метрика – количество уникальных средних, чем больше уникальных средних, тем разнообразней эффекты. С помощью данной метрики можно было определять крайности, например, что нейронная сеть переобучилась и генерирует из 20 эффектов только 3 уникальных средних. У данной метрики есть недостатки, которые присуще и самой метрике среднего, ведь у одного среднего значения могут быть совсем разные средние отклонения, и метрика не может этого учесть.

Статистические метрики. Среднее отклонение и коэффициент вариации. Для оценки такого массива эффектов, рассматривались поднаборы чисел для каждого числа, описывающего один параметр отдельно для всех эффектов. Например, брались в поднабор числа под индексом 10 описывающих размер частицы в каждом эффекте из основного набора, и оценивались разнообразие этого поднабора. Для оценки разнообразия рассматривались статистические метрики среднего отклонение и коэффициент вариации можно использовать среднее отклонение, но нельзя сказать с уверенностью чем больше отклонение, тем больше разнообразие, так как в наборе могут быть числа с большой разницей по модулю, но с низким разнообразием например [1, 1000, 1, 1000] имеют высокое отклонение и коэффициент вариации, а пример [1, 2, 1, 2] имеют низкие

показатели, но разнообразие технически такое же. По этим причинам эти метрики не использовались для оценки.

Количество уникальных чисел. Для расчёта - для каждого параметра выбирается шаг, при котором мы считаем числа разными, и считается количество уникальных чисел внутри каждого параметра, и простая сумма между всеми параметрами даёт метрику, чем выше это значение, тем больше уникальных эффектов генерирует сеть.

Распределение оценки дискриминатора. В рамках данной метрики использовался дискриминатор и его функция ошибки. На вход дискриминатору подавался набор эффектов из датасета и сгенерированных генератором, результаты оценивались функцией ошибки и распределялись от 0 до 1, по итогу получая условную метрику оценки вероятности реальности эффекта. Чем выше показатель, тем более вероятно, что эффект из датасета и чем ниже тем выше вероятность что эффект сгенерированный. Строя гистограмму распределения этого показателя можно оценить работу дискриминатора, увидеть крайние точки переобучения и предположить о разнообразии генерации, чем разнообразнее распределено, тем выше вероятность что генерация разнообразная. Пример гистограммы изображён на Рисунок 35.

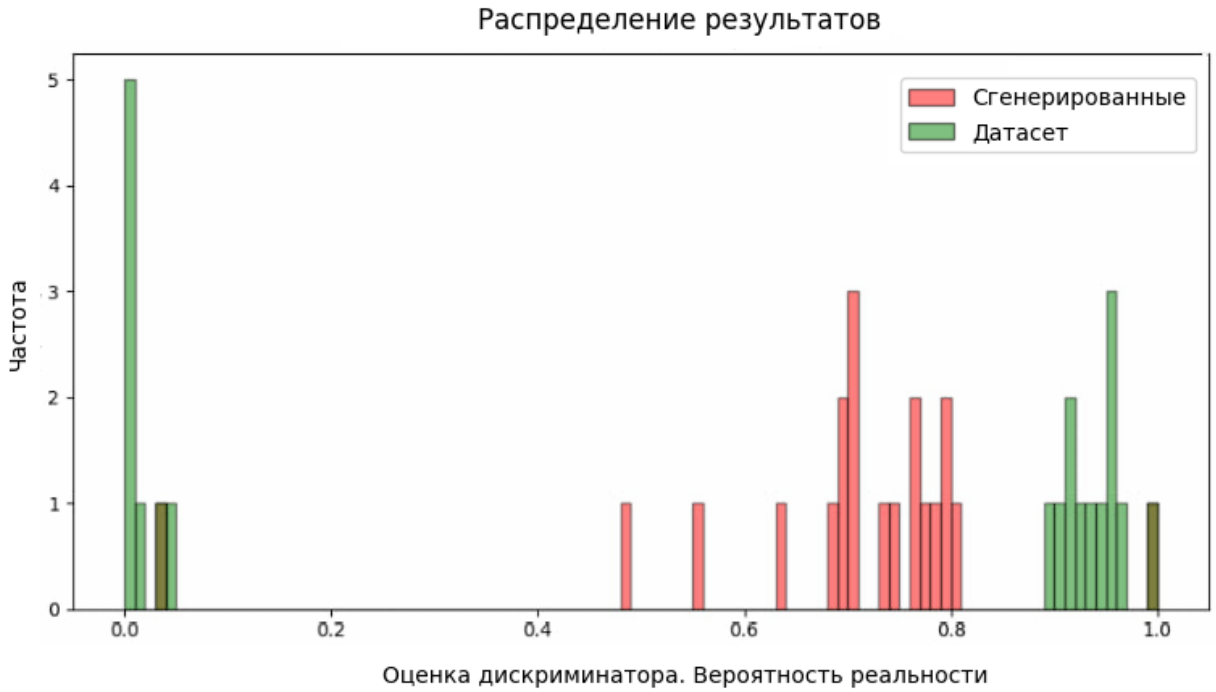


Рисунок 35 – Пример гистограммы распределения результатов оценки дискриминатора

Оценка классификации, разделения на группы. Для данной метрики используется стохастическое вложение соседей с t-распределением (t-SNE) [28] [29] – это алгоритм машинного обучения, который используется для отображения сложных данных в виде двумерного графика. Для этого на вход в алгоритм передаётся набор эффектов с указанными метками и алгоритм выдаёт набор точек графика X и Y, где каждая точка – это один эффект. Пример такого графика изображён на Рисунок 36. Значение осей для нас не имеют значения, мы рассматриваем расположение точек на этом графике. Можно обратить внимание что объекты из одной группы стремятся расположиться в отдельных областях.

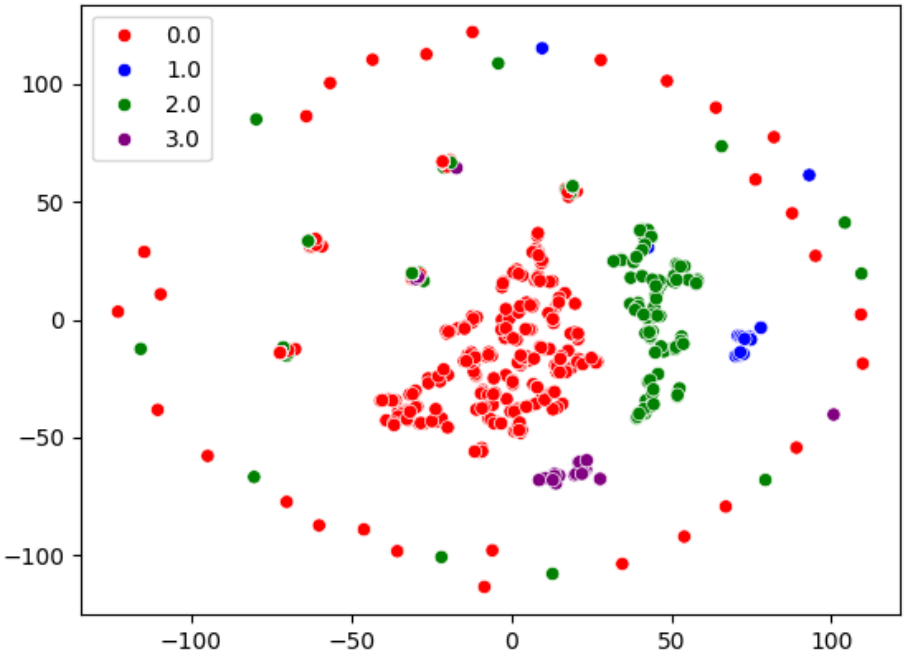


Рисунок 36 – Результат работы алгоритма t-SNE на сгенерированных данных

Далее для расчёта метрики используется другая нейронная сеть, которая на основе этих точек будет производить процесс классификации точек, и насколько точно нейронная сеть классификатор [30] сможет разделить эффекты на группы станет метрикой. Чем выше точность, тем лучше генератор научился по какому-либо признаку генерировать эффекты и выделять их в группы.

ЗАКЛЮЧЕНИЕ

В ходе данного исследования были рассмотрены ключевые аспекты визуальных эффектов (VFX) систем частиц в игровой индустрии и их важное влияние на итоговый вид игровых произведений. Было выявлено, что системы частиц могут значительно улучшить качество игровых визуальных эффектов, но их разработка требует значительных временных и финансовых затрат. Основываясь на этом, основной целью данного исследования было использование машинного обучения, в частности, нейронных сетей, для оптимизации процесса разработки VFX ассетов. В рамках исследования были определены следующие задачи:

– Анализ предметной области генеративно – состязательных нейронных сетей,

– Разработка теоретического решения инструмента для генерации ассетов партиклей для игровых движков,

– Практическая реализация инструмента,

– Реализация нейронной сети для генерации VFX ассетов,

– Проведение экспериментов с обучением нейронной сети.

Для решения задачи были выбраны генеративно-состязательные сети (GAN), которые представляют собой мощный инструмент в различных областях, таких как эффекты постпроцессинга, имитация голоса, 3D моделирование и генерация игровых уровней. Эти работы демонстрируют способность GAN к генерации контента, что открывает перспективы для дальнейшего исследования и расширения областей применения.

В исследовании был представлен и разработан пайплайн работы инструмента для генерации визуальных эффектов в игровых движках, который включает: инструмент для создания датасетов, обучение нейронной сети с использованием полученных датасетов и преобразование сгенерированных результатов нейронной сети в формат ассета для игрового движка. Такой подход отражает инновационный характер исследования, где

нейронные сети играют ключевую роль в автоматизации и улучшении процесса разработки визуальных эффектов.

В заключении, данное исследование предоставляет основу для дальнейшей работы над оптимизацией процесса создания VFX в играх и демонстрирует перспективы использования нейронных сетей в сфере создания визуальных эффектов.

Результаты решения и его исходные коды, текст, а также примеры с инструкциями опубликованы в репозитории на github.com, получить доступ к ним можно по ссылке https://github.com/LeGoLaZzZz/GAN_ParticlesGenerator.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Manrique, M. 2014. Blender for Animation and Film-Based Production. Boca Raton, FL: A K Peters/CRC Press.
2. Byrne B. The visual effects arsenal: VFX solutions for the independent filmmaker. – Routledge, 2012.
3. Reeves W. T. Particle systems—a technique for modeling a class of fuzzy objects //Seminal graphics: pioneering efforts that shaped the field. – 1998. – С. 203-220.
4. Reeves W. T., Blau R. Approximate and probabilistic algorithms for shading and rendering structured particle systems //ACM siggraph computer graphics. – 1985. – Т. 19. – №. 3. – С. 313-322.
5. Chiba N. et al. Two-dimensional visual simulation of flames, smoke and the spread of fire //The Journal of Visualization and Computer Animation. – 1994. – Т. 5. – №. 1. – С. 37-53.
6. Lotter R. Taking Blender to the Next Level: Implement advanced workflows such as geometry nodes, simulations, and motion tracking for Blender production pipelines. – Packt Publishing Ltd, 2022.
7. Raappana J. Great Particles and How to Make Them: Development of guidelines for the Unity Particle System. – 2018.
8. Wu Y. A new exploration based on unreal engine4 particle effects of unreal engine in 3d animation scenes //Int. J. Innov. Sci. Res. Technol. – 2021. – Т. 6. – С. 691-696.
9. Alzubaidi L. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions //Journal of big Data. – 2021. – Т. 8. – С. 1-74.
10. Lipton Z. C., Berkowitz J., Elkan C. A critical review of recurrent neural networks for sequence learning //arXiv preprint arXiv:1506.00019. – 2015.
11. Creswell A. et al. Generative adversarial networks: An overview //IEEE signal processing magazine. – 2018. – Т. 35. – №. 1. – С. 53-65.

12. Karp R., Swiderska-Chadaj Z. Automatic generation of graphical game assets using GAN //2021 7th International Conference on Computer Technology Applications. – 2021. – С. 7-12.
13. Tschannen M., Bachem O., Lucic M. Recent advances in autoencoder-based representation learning //arXiv preprint arXiv:1812.05069. – 2018.
14. Li P., Pei Y., Li J. A comprehensive survey on design and application of autoencoder in deep learning //Applied Soft Computing. – 2023. – Т. 138. – С. 110176.
15. Ren L, Song Y. AOGAN: A generative adversarial network for screen space ambient occlusion. Computational Visual Media, 2022, 8(3): 483-494. URL: https://doi.org/10.1007/s41095-021-0248-2 (дата обращения: 15.12.2022).
16. Ronneberger O., Fischer P., Brox T. U-net: Convolutional networks for biomedical image segmentation //International Conference on Medical image computing and computer-assisted intervention. – Springer, Cham, 2015. – С. 234-241.
17. Isola P. et al. Image-to-image translation with conditional adversarial networks //Proceedings of the IEEE conference on computer vision and pattern recognition. – 2017. – С. 1125-1134.
18. H. Phan et al., "Self-Attention Generative Adversarial Network for Speech Enhancement," ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 7103-7107.
19. Gao Y., Singh R., Raj B. Voice impersonation using generative adversarial networks //2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2018. – С. 2506-2510.
20. Wu J. et al. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling //Advances in neural information processing systems. – 2016. – Т. 29.
21. Kodali N. et al. On convergence and stability of gans //arXiv preprint arXiv:1705.07215. – 2017.

22. Yu Y. et al. A review of recurrent neural networks: LSTM cells and network architectures //Neural computation. – 2019. – Т. 31. – №. 7. – С. 1235-1270.
23. Daniel Schroeder, Howard J. Hamilton, 2014, Desirable Elements for a Particle System Interface https://doi.org/10.1155/2014/623809 (дата обращения: 13.12.2023).
24. McAllister D. K. The design of an API for particle systems //University of North Carolina Technical Report TR 00-007. – 2000.
25. Mirza M., Osindero S. Conditional generative adversarial nets //arXiv preprint arXiv:1411.1784. – 2014.
26. Antipov G., Baccouche M., Dugelay J. L. Face aging with conditional generative adversarial networks //2017 IEEE international conference on image processing (ICIP). – IEEE, 2017. – С. 2089-2093.
27. Tang J. et al. Line: Large-scale information network embedding //Proceedings of the 24th international conference on world wide web. – 2015. – С. 1067-1077
28. Van der Maaten L., Hinton G. Visualizing data using t-SNE //Journal of machine learning research. – 2008. – Т. 9. – №. 11.
29. Albacete Zapata A. An in-depth review of t-SNE with applications. – 2019.
30. Dreiseitl S., Ohno-Machado L. Logistic regression and artificial neural network classification models: a methodology review //Journal of biomedical informatics. – 2002. – Т. 35. – №. 5-6. – С. 352-359.

Электронные ресурсы:

1. Game Level Generation Using Neural Networks [Электронный ресурс]. – 2018 – URL: https://www.gamedeveloper.com/programming/game-level-generation-using-neural-networks (дата обращения 13.12.2023).

2. Unity Asset Store 52 Special Effects Pack [Электронный ресурс] https://assetstore.unity.com/packages/vfx/particles/spells/52-special-effects-pack-10419 (дата обращения 17.12.2023).