

Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Кафедра «школы бакалавриата (школа)»

Оценка работы зачтено
Руководитель от УрФУ Зайнуллина А.М.


Тема задания на практику

Практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

ОТЧЕТ

Вид практики Учебная практика

Тип практики Практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

Руководитель практики от предприятия (организации) Зайнуллина А.М. 
ФИО руководителя Подпись

Студент Ярков Т.М.
ФИО студента

Специальность (направление подготовки) 10.05.01 Компьютерная безопасность

Группа МЕН-201015

Екатеринбург 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
ОСНОВНАЯ ЧАСТЬ.....	5
1 Формулировка задачи.....	5
2 Теория.....	5
2.0 Основные определения	5
2.1.1 Принципы Deep learning.....	5
2.1.2 Принципы эволюционных алгоритмов.....	6
2.2 Алгоритм NEAT.....	8
2.2.1 Принципы работы.....	8
2.2.2 Процесс эволюции нейронных сетей с использованием NEAT.....	9
2.2.3 Возможности NEAT в решении задач в области игр.....	11
2.3 NEAT-Python.....	13
2.3.0 Определения.....	13
2.3.1 Введение в NEAT-Python.....	14
2.3.2 Настройка файла config.txt.....	15
3 Реализация игры на Python.....	24
3.1 Описание проекта.....	24
3.2 Скриншоты игры	38
ЗАКЛЮЧЕНИЕ.....	40
СПИСОК ЛИТЕРАТУРЫ.....	42
ПРОЛОЖЕНИЯ.....	43

ВВЕДЕНИЕ

Глубокое обучение (Deep learning) и эволюционные алгоритмы представляют собой две важные области искусственного интеллекта, которые получили значительное внимание в последнее десятилетие. Глубокие нейронные сети, благодаря своей способности извлекать сложные закономерности из больших объемов данных, привели к революционным достижениям в области обработки изображений, распознавания речи, естественного языка и других задач машинного обучения. С другой стороны, эволюционные алгоритмы, вдохновленные биологическим процессом эволюции, предлагают эффективные методы оптимизации и поиска решений в сложных пространствах параметров.

Актуальность данной работы заключается в том, что в современном мире глубокое обучение (Deep learning) является одной из наиболее активно развивающихся областей искусственного интеллекта. Применение глубоких нейронных сетей привело к значительному прогрессу в различных областях, включая компьютерное зрение, обработку естественного языка, робототехнику и автономную навигацию. Однако, создание эффективных архитектур нейронных сетей все еще остается сложной задачей. В этом контексте, использование эволюционных алгоритмов, таких как NEAT, для автоматического проектирования нейронных сетей представляет собой актуальное исследовательское направление.

Алгоритм NEAT (NeuroEvolution of Augmenting Topologies) объединяет глубокое обучение и эволюционные методы с целью создания и оптимизации нейронных сетей. NEAT предлагает новаторский подход к автоматическому проектированию и эволюции архитектур нейронных сетей, обеспечивая возможность создания сложных и гибких моделей, способных адаптироваться к различным задачам и условиям.

В рамках данного исследования будут рассмотрены основные NEAT. Будет исследован процесс эволюции нейронных сетей с использованием NEAT, а также методы оценки и селекции лучших решений. Кроме того, будут рассмотрены практические примеры применения NEAT.

Цель данной работы заключается в использовании NEAT в качестве мощного инструмента для автоматического проектирования нейронных сетей и решения сложных задач машинного обучения и направлена на обеспечение более глубокого понимания принципов NEAT и его потенциала в создании игр.

В соответствии с поставленной целью в работе определены задачи:

1. Изучить теорию.
2. Изучить библиотеку NEAT-Python.
3. Выполнить программную реализацию игры с использованием алгоритма NEAT.

Объектом исследования является алгоритм NEAT.

Предметом исследования являются методы и подходы, связанные с применением NEAT для создания и оптимизации архитектур нейронных сетей. Исследование будет включать процесс эволюции нейронных сетей с использованием NEAT, методы оценки и селекции лучших решений, а также применение NEAT в играх.

Курсовая работа содержит введение, основную часть, заключение, список литературы, приложения.

ОСНОВНАЯ ЧАСТЬ

1 Формулировка задачи

Необходимо исследовать алгоритм NEAT(NeuroEvolution of Augmenting Topologies) в контексте Deep learning. Изучить принцип и механизм работы NEAT, проанализировать его алгоритмические особенности и возможности в создании эффективных нейронных сетей.

Задачи:

1. Изучить основные принципы глубокого обучения и эволюционных алгоритмов, включая основные понятия и методы обоих подходов.
2. Изучить алгоритм NEAT в деталях, включая его историю, основные компоненты и принципы работы.
3. Рассмотреть процесс эволюции нейронных сетей с использованием NEAT, включая методы мутации и селекции, а также принципы формирования новых архитектур.
4. Проанализировать возможности NEAT в решении задач глубокого обучения, включая его применение в области игр.
5. Реализовать игру с применением алгоритма NEAT.

2 Теория

2.0 Основные определения

Deep learning(глубокое обучение) — совокупность методов машинного обучения (с учителем, с частичным привлечением учителя, без учителя, с подкреплением), основанных на обучении представлениям, а не специализированных алгоритмах под конкретные задачи.

2.1.1 Принципы Deep learning

1. Иерархическая структура: Основной принцип глубокого обучения заключается в использовании иерархической структуры нейронных сетей. Глубокая нейронная сеть состоит из нескольких слоев, где каждый

слой обрабатывает и извлекает все более абстрактные и сложные признаки из входных данных. Благодаря иерархической структуре нейронные сети могут автоматически изучать сложные закономерности и представлять данные на различных уровнях абстракции.

2. Обратное распространение ошибки: В глубоком обучении широко применяется алгоритм обратного распространения ошибки. Он основан на вычислении градиента функции потерь по весам нейронной сети. Путем последовательного обновления весов на основе градиента ошибки сеть постепенно настраивается на оптимальные значения, что позволяет ей улучшать свою способность к предсказанию или классификации данных.

3. Автоматическое извлечение признаков: Глубокие нейронные сети способны автоматически извлекать репрезентативные признаки из данных без явного задания правил или характеристик. Вместо того, чтобы ручным образом определять признаки, глубокое обучение позволяет нейронной сети самостоятельно находить и изучать наиболее информативные признаки, что делает его более гибким и способным работать с различными типами данных.

4. Масштабируемость. Еще одним принципом глубокого обучения является его масштабируемость. Глубокие нейронные сети могут иметь большое количество слоев и огромное количество параметров. Благодаря этой масштабируемости глубокое обучение способно моделировать и обрабатывать сложные данные с большой размерностью, такие как изображения, аудио или текст.

2.1.2 Принципы эволюционных алгоритмов

1. Наследование и изменчивость. Одним из ключевых принципов эволюционных алгоритмов является наследование и изменчивость. Это

означает, что в каждом поколении алгоритма создаются новые особи, которые наследуют свойства и характеристики от предыдущего поколения, но также вносят в них некоторые изменения или вариации. Это позволяет внести разнообразие в популяцию и исследовать различные варианты решений.

2. Отбор по приспособленности. Эволюционные алгоритмы основаны на концепции естественного отбора, где особи с лучшей приспособленностью имеют больше шансов на выживание и передачу своих генетических характеристик следующему поколению. Это позволяет алгоритму постепенно улучшать решения и сходиться к оптимальному или приближенно оптимальному решению задачи.

3. Мутация и скрещивание. Два основных механизма изменения генетического материала в эволюционных алгоритмах - это мутация и скрещивание. Мутация представляет собой случайные изменения в генетическом коде особи, которые вносят вариации и новые свойства. Скрещивание, с другой стороны, комбинирует генетический материал от двух или более особей, чтобы создать новые потомки с комбинированными свойствами. Эти механизмы позволяют вносить разнообразие и исследовать пространство возможных решений.

4. Оценка приспособленности. Для эффективного отбора особей необходимо оценивать их приспособленность к решаемой задаче. Оценка приспособленности происходит на основе некоторой метрики или функции, которая определяет, насколько хорошо особь справляется с задачей. Оценка приспособленности позволяет выделить лучшие особи и провести отбор для формирования следующего поколения.

5. Итеративность. Эволюционные алгоритмы работают по принципу итераций, где каждая итерация представляет собой одно поколение. В каждой итерации особи создаются, оцениваются, происходят мутации и скрещивания, а затем выбираются лучшие особи для следующего поколения. Процесс итеративно повторяется до достижения определенного критерия останова или сходимости.

2.2 Алгоритм NEAT

2.2.1 Принципы работы

NEAT (NeuroEvolution of Augmenting Topologies) - это алгоритм эволюционного обучения, который используется для эволюции и оптимизации нейронных сетей. NEAT был разработан Кенни Стэнли и Рэнди Шербином в 2002 году и с тех пор стал популярным инструментом в области искусственного интеллекта и генетического программирования.

Основная идея NEAT заключается в том, что алгоритм самостоятельно эволюционирует нейронные сети, начиная с простых структур, и постепенно увеличивает их сложность и архитектуру. Одной из ключевых особенностей NEAT является поддержка развивающихся топологий, то есть возможность добавления и удаления нейронов и связей в процессе эволюции.

Процесс эволюции в NEAT происходит поэтапно. В начале алгоритма создается начальная популяция нейронных сетей с простыми структурами, состоящими из небольшого числа нейронов и связей. Затем особи в популяции оцениваются по мере их приспособленности к решаемой задаче.

Далее происходит процесс размножения и создания новых поколений. NEAT использует механизмы мутации и скрещивания для изменения генетического материала нейронных сетей. Мутации могут включать добавление новых нейронов, добавление новых связей, изменение весов

связей и другие модификации. Скрещивание происходит путем комбинирования генетического материала двух родительских нейронных сетей.

Особи в новом поколении оцениваются на приспособленность, и наиболее приспособленные особи продолжают в следующее поколение. Таким образом, через итерации процесса эволюции, нейронные сети становятся все более сложными и оптимизированными для решения задачи.

Преимуществом NEAT является его способность эволюционировать нейронные сети с самоорганизующейся архитектурой, позволяющей изучать сложные связи и оптимизировать структуру сети. Также NEAT обладает способностью обходить проблему исчезающего градиента, которая может возникать при обучении глубоких нейронных сетей.

NEAT стал популярным инструментом в области обучения нейронных сетей, особенно для задач, где требуется эволюционная оптимизация структуры сети, а также для создания интеллектуальных агентов в симуляциях и играх.

2.2.2 Процесс эволюции нейронных сетей с использованием NEAT

Процесс эволюции нейронных сетей с использованием алгоритма NEAT (NeuroEvolution of Augmenting Topologies) включает несколько этапов, таких как инициализация популяции, оценка приспособленности, операторы мутации и скрещивания, а также формирование новых архитектур.

Рассмотрим эти этапы подробнее:

1. Инициализация популяции. В начале процесса эволюции NEAT создает начальную популяцию нейронных сетей с простыми структурами, состоящими из небольшого числа нейронов и связей. Инициализация

может происходить случайным образом или с использованием некоторых эвристик.

2. Оценка приспособленности. После инициализации каждая нейронная сеть в популяции оценивается на основе ее приспособленности к решаемой задаче. Приспособленность может быть определена с использованием целевой функции, которая измеряет эффективность сети в решении задачи. Чем лучше сеть справляется с задачей, тем выше ее приспособленность.
3. Мутация. Мутация является одним из ключевых операторов в NEAT и позволяет вносить изменения в генетический материал нейронной сети. Во время мутации могут происходить следующие изменения:
 - Добавление нейрона. Мутация может добавить новый нейрон в существующую сеть. Новый нейрон может быть связан с другими нейронами, создавая новые пути передачи информации.
 - Добавление связи. Мутация может добавить новую связь между двумя существующими нейронами в сети. Новая связь представляет собой дополнительный путь передачи сигнала.
 - Изменение весов. Мутация может изменить веса существующих связей в сети. Изменение весов позволяет алгоритму исследовать пространство возможных решений и оптимизировать производительность сети.
4. Скрещивание. Скрещивание (кроссовер) является еще одним оператором в NEAT и позволяет комбинировать генетический материал из двух родительских нейронных сетей для создания потомков. При скрещивании случайным образом выбираются связи и нейроны из обоих родителей, и потомки получают комбинацию этих

связей и нейронов. Скрещивание помогает сохранять полезные характеристики обоих родителей и исследовать новые комбинации в архитектуре сети.

5. **Формирование новых архитектур:** В NEAT новые архитектуры формируются с использованием принципа сохранения истории. Каждая связь и нейрон в сети имеют свой уникальный идентификатор, что позволяет отслеживать историю и эволюцию структуры сети. Новые архитектуры могут быть сформированы путем добавления новых нейронов или связей в процессе мутации и скрещивания.

2.2.3 Возможности NEAT в решении задач в области игр

NEAT (NeuroEvolution of Augmenting Topologies) является мощным инструментом для решения задач в области игр и создания интеллектуальных агентов. Вот несколько возможностей NEAT в этой области:

1. **Автоматическое обучение.** NEAT позволяет автоматически эволюционировать нейронные сети для игровых задач. Агенты обучаются самостоятельно, проходя через множество поколений, и постепенно улучшают свои навыки и стратегии. Это особенно полезно в случаях, когда задача сложна или неизвестна заранее, и требуется адаптивное исследование пространства решений.
2. **Приспособление к изменяющимся условиям.** NEAT позволяет агентам адаптироваться к изменяющимся условиям в игре. Благодаря возможности добавления новых нейронов и связей в процессе эволюции агенты могут изменять свою архитектуру и стратегию в ответ на новые вызовы и ситуации в игре.

3. Решение сложных стратегических задач. NEAT способен эволюционировать нейронные сети с глубокой и сложной структурой, что позволяет агентам разрабатывать сложные стратегии в играх. Агенты могут находить оптимальные пути, предсказывать действия противников и адаптироваться к различным игровым сценариям.
4. Исследование пространства решений. NEAT позволяет исследовать пространство решений в игре, искать оптимальные стратегии и находить неожиданные подходы к решению задач. Эволюционный процесс позволяет агентам исследовать различные комбинации нейронов, связей и весов, что может привести к открытию новых и эффективных способов решения задач.
5. Подходит для различных типов игр. NEAT может быть применен к различным типам игр, включая стратегические игры, головоломки, аркады и другие. Алгоритм не зависит от специфики игры и может быть адаптирован к различным средам и правилам игры.

В целом, NEAT предоставляет мощный инструмент для автоматического обучения интеллектуальных агентов в играх. Он позволяет развивать сложные стратегии, адаптироваться к изменяющимся условиям и исследовать пространство решений, что делает его ценным инструментом для разработки интеллектуальных систем в области игр. Исходя из сказанного выше, была выбрана библиотека NEAT-Python для применения алгоритма NEAT в реализации игры. С помощью данной библиотеки можно создавать и эволюционировать нейронные сети с заданными настройками и параметрами.

2.3 NEAT-Python

2.3.0 Определения

Ген (gene)

Информация, кодирующая конкретный аспект (узел или соединение) фенотипа нейронной сети. Содержит несколько атрибутов, различающихся в зависимости от типа гена. Примеры классов генов включают `genes.DefaultNodeGene`, `genes.DefaultConnectionGene` и `iznn.IZNodeGene`; все они являются подклассами `genes.BaseGene`.

Соединение (connection)

Они соединяют узлы между собой и дают начало сети в термине "нейронная сеть". Для нерекуррентных (непосредственно рекуррентных) соединений они эквивалентны биологическим синапсам. Соединения имеют два атрибута, их вес и то, включены они или нет; оба определяются их геном. Пример генного класса для соединений можно увидеть в `genes.DefaultConnectionGene`.

Узел (node)

Также известен как нейрон (как в нейронной сети). Они бывают трех типов: входные, скрытые и выходные. Узлы имеют один или несколько атрибутов, например, функцию активации; все они определяются их геном. Классы узловых генов включают `genes.DefaultNodeGene` и `iznn.IZNodeGene`.

Атрибуты (attributes)

Это свойства узла (например, его функция активации) или соединения (например, включено оно или нет), определяемые связанным с ним геном.

Функция активации (Activation Function)

Функция активации решает, должен ли нейрон быть активирован или нет. Это означает, что она будет решать, важен или нет вход нейрона в сеть в

процессе прогнозирования, используя более простые математические операции.

2.3.1 Введение в NEAT-Python

NEAT-Python - это популярная библиотека на языке программирования Python, предназначенная для реализации алгоритма NEAT (NeuroEvolution of Augmenting Topologies). NEAT-Python предоставляет удобный интерфейс и множество функций, которые позволяют разработчикам создавать, эволюционировать и обучать нейронные сети с использованием NEAT.

В текущей реализации NEAT-Python поддерживается популяция индивидуальных геномов. Каждый геном содержит два набора генов, которые описывают, как построить искусственную нейронную сеть:

- Гены узлов, каждый из которых определяет один нейрон.
- Гены соединений, каждый из которых определяет одно соединение между нейронами.

Для эволюции пользователь должен предоставить fitness-функцию, которая вычисляет единственное вещественное число, указывающее на качество отдельного генома: лучшая способность решить проблему означает более высокий балл. Алгоритм проходит через заданное пользователем число поколений, причем каждое поколение создается путем размножения и мутации наиболее приспособленных особей предыдущего поколения.

Операции размножения и мутации могут добавлять узлы и/или связи в геномы, поэтому по мере работы алгоритма геномы (и создаваемые ими нейронные сети) могут становиться все более сложными. По достижении заданного числа поколений или когда хотя бы одна особь (для функции

критерия пригодности \max ; другие настраиваются) превысит заданный пользователем порог пригодности, алгоритм завершается.

Одна из трудностей в этой установке связана с реализацией кроссинговера - как провести кроссинговер между двумя сетями с различной структурой? NEAT решает эту проблему, отслеживая происхождение узлов с помощью идентификационного номера (для каждого дополнительного узла генерируются новые, более высокие номера). Узлы, происходящие от общего предка (гомологичные), подбираются для кроссинговера, а соединения подбираются, если узлы, которые они соединяют, имеют общих предков.

Другая потенциальная трудность заключается в том, что структурные мутации - в отличие от мутаций, например, в весах связей - такие как добавление узла или связи, могут, хотя и быть многообещающими в будущем, быть разрушительными в краткосрочной перспективе (пока не будут отлажены менее разрушительными мутациями). NEAT решает эту проблему путем разделения геномов на виды, которые имеют близкое геномное расстояние из-за сходства, затем конкуренция становится наиболее интенсивной внутри вида, а не между видами (разделение приспособленности). Как измеряется геномное расстояние? Используется комбинация количества негомологичных узлов и связей с мерами того, насколько гомологичные узлы и связи разошлись с момента их общего происхождения.

2.3.2 Настройка файла config.txt

Файл config.txt в NEAT-Python является конфигурационным файлом, который позволяет настраивать параметры алгоритма NEAT для вашей конкретной задачи. В этом файле можно определить различные параметры, включая размер популяции, вероятности мутаций и скрещиваний, методы

селекции, функции оценки приспособленности и другие. Конфигурационный файл позволяет детально настроить параметры NEAT-Python для конкретной задачи. Также можете изменять значения параметров, экспериментировать с различными настройками и адаптировать алгоритм под ваши потребности.

Параметры, используемые в проекте для настройки алгоритма:

fitness_criterion

Функция, используемая для вычисления критерия прекращения по набору приспособленностей генома. Допустимые значения: min, max, и mean

fitness_threshold

Когда фитнес, вычисленный fitness_criterion, соответствует или превышает этот порог, процесс эволюции завершается с вызовом метода found_solution любого зарегистрированного класса отчетов.

pop_size

Количество особей в каждом поколении.

reset_on_extinction

Если это значение равно True, то когда все виды одновременно вымрут из-за стагнации, будет создана новая случайная популяция. Если False, будет выброшено исключение CompleteExtinctionException.

activation_default

Атрибут функции активации по умолчанию, назначаемый новым узлам. Если не задано ни одного, или указано "random", то один из вариантов activation_options будет выбран случайным образом.

activation_mutate_rate

Вероятность того, что мутация заменит агрегационную функцию узла на случайно определенный член `aggregation_options`. Допустимые значения лежат в пределах $[0.0, 1.0]$.

activation_options

Разделенный пробелами список функций активации, которые могут использоваться узлами. По умолчанию это сигмоид. Доступные встроенные функции можно найти в разделе Обзор встроенных функций активации; можно добавить больше, как описано в разделе «Настройка поведения».

aggregation_default

Атрибут функции агрегации по умолчанию, назначаемый новым узлам. Если не задано ни одного, или указано "random", то один из параметров `aggregation_options` будет выбран случайным образом.

aggregation_mutate_rate

Вероятность того, что мутация заменит функцию агрегации узла randomly-determined элементом `aggregation_options`. Допустимые значения находятся в $[0,0, 1,0]$.

aggregation_options

Список функций агрегирования, которые могут использоваться узлами, разделенный пробелами. По умолчанию используется значение "sum". Доступные функции (определенные в агрегации): sum, product, min, max, mean, median и maxabs (которая возвращает входное значение с наибольшим абсолютным значением; возвращаемое значение может быть положительным или отрицательным). Новые функции агрегирования могут быть определены аналогично новым функциям активации. (Обратите внимание, что функция должна принимать список или другую итерируемую величину; здесь может пригодиться функция reduce, как в агрегации).

bias_init_mean

Среднее значение нормального/гауссова распределения, если оно используется для выбора значений атрибутов смещения для новых узлов.

bias_init_stdev

Стандартное отклонение нормального/гауссовского распределения, оно используется для выбора значений смещения для новых узлов.

bias_max_value

Максимально допустимое значение смещения. Смещения, превышающие это значение, будут ужиматься до этого значения.

bias_min_value

Минимально допустимое значение смещения. Смещения ниже этого значения будут ужиматься до этого значения.

bias_mutate_power

Стандартное отклонение нуль-центрированного нормального/гауссова распределения, из которого берется мутация значения смещения.

bias_mutate_rate

Вероятность того, что мутация изменит смещение узла путем добавления случайного значения.

bias_replace_rate

Вероятность того, что мутация заменит смещение узла на вновь выбранное случайное значение (как если бы это был новый узел).

compatibility_disjoint_coefficient

Коэффициент вклада несовпадающих и избыточных количеств генов в геномное расстояние.

compatibility_weight_coefficient

Коэффициент для вклада каждого веса, смещения или множителя ответа в геномное расстояние (для гомологичных узлов или соединений). Этот коэффициент также используется в качестве значения для добавления различий в функциях активации, функциях агрегирования или включенном/выключенном состоянии.

conn_add_prob

Вероятность того, что мутация добавит соединение между существующими узлами. Допустимые значения находятся в пределах [0.0, 1.0].

conn_delete_prob

Вероятность того, что мутация удалит существующее соединение. Допустимые значения находятся в пределах [0.0, 1.0].

enabled_default

Включенный по умолчанию атрибут вновь созданных подключений. Допустимые значения True и False.

enabled_mutate_rate

Вероятность того, что мутация (с вероятностью 50/50 True или False) активирует статус соединения. Допустимые значения находятся в [0,0, 1,0].

feed_forward

Если значение этого параметра равно True, то создаваемым сетям не будет разрешено иметь рекуррентные связи (они будут feedforward). В противном случае они могут быть (но не обязаны быть) рекуррентными.

node_add_prob

Вероятность того, что мутация добавит новый узел (по сути, заменит существующее соединение, включенное состояние которого будет установлено в False). Допустимые значения находятся в диапазоне [0.0, 1.0].

node_delete_prob

Вероятность того, что мутация удалит существующий узел (и все связи с ним). Допустимые значения находятся в пределах [0.0, 1.0].

num_hidden

Количество скрытых узлов, добавляемых к каждому геному в начальной популяции.

num_inputs

Количество входных узлов, через которые сеть получает входные данные.

num_outputs

Количество выходных узлов, на которые сеть подает выходные данные.

response_init_mean

Среднее значение нормального/гауссовского распределения, если оно используется для выбора значений атрибутов множителя отклика для новых узлов.

response_init_stdev

Стандартное отклонение нормального/гауссова распределения, если оно используется для выбора множителей отклика для новых узлов.

response_max_value

Максимально допустимый множитель отклика. Множители отклика, превышающие это значение, будут ужиматься до этого значения.

response_min_value

Минимально допустимый множитель отклика. Множители отклика ниже этого значения будут ужиматься до этого значения.

response_mutate_power

Стандартное отклонение нуль-центрированного нормального/гауссова распределения, из которого берется мутация множителя ответа.

response_mutate_rate

Вероятность того, что мутация изменит множитель ответа узла путем добавления случайного значения.

response_replace_rate

Вероятность того, что мутация заменит множитель ответа узла на вновь выбранное случайное значение (как если бы это был новый узел).

weight_init_mean

Среднее значение нормального/гауссова распределения, используемого для выбора значений весовых атрибутов для новых соединений.

weight_init_stdev

Стандартное отклонение нормального/гауссова распределения, используемого для выбора значений веса для новых соединений.

weight_max_value

Максимально допустимое значение веса. Вес, превышающий это значение, будет ужиматься до этого значения.

weight_min_value

Минимально допустимое значение веса. Вес, не достигающий этого значения, будет зажиматься до этого значения.

weight_mutate_power

Стандартное отклонение нуль-центрированного нормального/гауссова распределения, из которого берется мутация значения веса.

weight_mutate_rate

Вероятность того, что мутация изменит вес соединения, добавив случайное значение.

weight_replace_rate

Вероятность того, что мутация заменит вес соединения на вновь выбранное случайное значение (как если бы это было новое соединение).

compatibility_threshold

Особи, геномное расстояние которых меньше этого порога, считаются принадлежащими к одному и тому же виду.

pecies_fitness_func

Функция, используемая для вычисления приспособленности вида. По умолчанию используется значение `mean`. Допустимые значения: `max`, `min`, `mean`, и `median`

`max_stagnation`

Виды, которые не показали улучшения в течение более чем этого количества поколений, будут считаться застойными и будут удалены. По умолчанию это 15.

`species_elitism`

Количество видов, которые будут защищены от стагнации; в основном предназначен для предотвращения полного вымирания, вызванного стагнацией всех видов до появления новых видов. Например, значение параметра `species_elitism` равное 3, предотвратит удаление трех видов с наивысшим уровнем приспособленности за стагнацию, независимо от того, сколько времени они не демонстрировали улучшения. По умолчанию это 0.

`elitism`

Количество наиболее подходящих особей каждого вида, которые будут сохраняться в неизменном виде от одного поколения к другому. По умолчанию это значение равно 0.

`survival_threshold`

Доля для каждого вида, которой разрешено размножаться в каждом поколении. По умолчанию это значение равно 0,2.

Атрибут функции активации по умолчанию, назначаемый новым узлам. Если не задано ни одного, или указано "random", то один из вариантов `activation_options` будет выбран случайным образом.

3 Реализация игры на Python

3.1 Описание проекта

Проект состоит из 5 файлов с расширением .py и 10 файлов с изображениями.

- 1) car.py - класс по созданию модели машины
- 2) draw_car.py – файл с методами по созданию машины и отрисовке их на карте
- 3) draw_radar.py – класс по отрисовке радара
- 4) start_window.py – класс, создающий начальное окно в проекте, в котором пользователь выбирает карту
- 5) game_neat – класс по управлению машиной алгоритмом NEAT

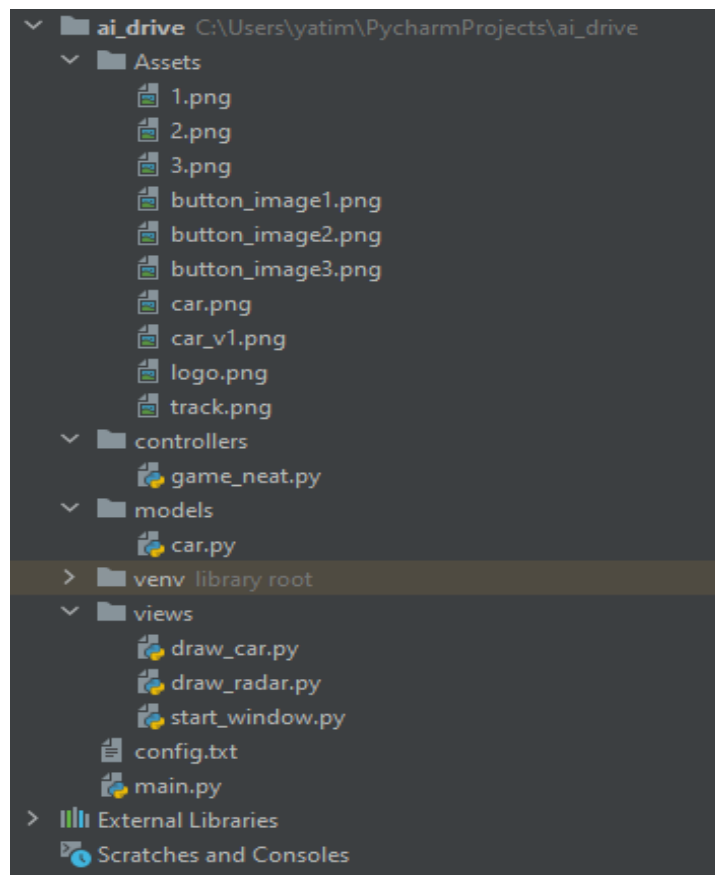


Рисунок 1.1 – Структура проекта

Рассмотрим каждый файл и разберем методы(Все методы имеют поясняющие docstrings):

- 1) car.py

Имеет 7 функций, которые необходимо рассмотреть


```

2 usages  yatim-dev
9  class Car(pygame.sprite.Sprite):
10     """Класс по созданию модели машины"""
11     yatim-dev
12     def __init__(self, window):
13         super().__init__()
14         self.original_image = pygame.image.load(os.path.join("Assets", "car.png"))
15         self.image = self.original_image
16         self.screen = window
17         self.rect = self.image.get_rect(center=(600, 820))
18         self.alive = True
19         self.vel_vector = pygame.math.Vector2(0.8, 0)
20         self.angle = 0
21         self.rotation_vel = 5
22         self.direction = 0
23         self.radars = []
24         self.draw_radar = DrawRadar()

```

Рисунок 1.2 – Инициализатор класса Car

```

1 usage (1 dynamic)  yatim-dev
def update(self):
    """Обновление машины с радаром, углом поворота, радаром"""
    self.radars.clear()
    self.drive()
    self.rotate()
    for radar_angle in (-60, -30, 0, 30, 60):
        self.radar(radar_angle)
    self.collision()
    self.data()

```

Рисунок 1.2 – метод update класса Car

```

1 usage  yatim-dev
def drive(self):
    """Метод, ответственный за скорость машины"""
    self.rect.center += self.vel_vector * 6

```

Рисунок 1.3 – метод drive класса Car

```

1 usage  🐦 yatim-dev
def rotate(self):
    """Метод для поворота машины"""
    if self.direction == 1:
        self.angle -= self.rotation_vel
        self.vel_vector.rotate_ip(self.rotation_vel)
    if self.direction == -1:
        self.angle += self.rotation_vel
        self.vel_vector.rotate_ip(-self.rotation_vel)

    self.image = pygame.transform.rotozoom(self.original_image, self.angle, 0.1)
    self.rect = self.image.get_rect(center=self.rect.center)

```

Рисунок 1.4 – метод rotate класса Car

```

1 usage  🐦 yatim-dev *
def radar(self, radar_angle):
    """Метод для работы с радаром"""
    length = 0
    x = int(self.rect.center[0])
    y = int(self.rect.center[1])

    length, x, y = self.draw_radar.scanner(self.screen, self.rect,
                                           self.angle, x, y, radar_angle, length)

    self.draw_radar.draw(self.screen, self.rect, x, y)

    # Расстояние между центром машины и верхушкой радара
    dist = int(math.sqrt(math.pow(self.rect.center[0] - x, 2)
                          + math.pow(self.rect.center[1] - y, 2)))

    self.radars.append([radar_angle, dist])

```

Рисунок 1.5 – метод radar класса Car

```

2 usages (1 dynamic)  🐦 yatim-dev
def data(self):
    """Сбор данных с радаров для NEAT"""
    data = [0 for _ in range(5)]
    for i, radar in enumerate(self.radars):
        data[i] = int(radar[1])
    return data

```

Рисунок 1.6 – метод data класса Car

```

1 usage  yatim-dev *
def collision(self):
    """Метод для создания точек коллизии"""
    length = 40
    collision_point_right = [int(self.rect.center[0] +
                               math.cos(math.radians(self.angle + 18)) * length),
                             int(self.rect.center[1] -
                               math.sin(math.radians(self.angle + 18)) * length)]
    collision_point_left = [int(self.rect.center[0] +
                               math.cos(math.radians(self.angle - 18)) * length),
                             int(self.rect.center[1] -
                               math.sin(math.radians(self.angle - 18)) * length)]

    # Умереть при столкновении
    if self.screen.get_at(collision_point_right) == pygame.Color(2, 105, 31, 255) \
        or self.screen.get_at(collision_point_left) == pygame.Color(2, 105, 31, 255):
        self.alive = False

    # Рисуем точки столкновения
    self.draw_radar.draw_col_points(self.screen, collision_point_right, collision_point_left)

```

Рисунок 1.7 – метод collision класса Car

2) draw_car.py

Имеет 2 функции, которые необходимо рассмотреть

```

1 usage  yatim-dev
def spawn_car(screen):
    """Метод по созданию машин

    :parameter
    screen : pygame.display
             окно, в котором происходит отрисовка

    :return
    GroupSingle
             группа спрайтов
    """
    return pygame.sprite.GroupSingle(Car(screen))

```

Рисунок 2.1 – метод spawn_car файла draw_car.py

```

1 usage  yatim-dev
def update_car(cars, screen):
    """Метод по отрисовки машины на карте

    :parameter
    cars : list
        коллекция машин
    screen : pygame.display
        окно, в котором происходит отрисовка
    """
    for car in cars:
        car.draw(screen)
        car.update()
    pygame.display.update()

```

Рисунок 2.2 – метод update_car файла draw_car.py

3) draw_radar.py

Имеет 3 функции, которые необходимо рассмотреть

```

1 usage  yatim-dev
def scanner(self, screen, rect, angle, x, y, radar_angle, length):
    """Метод по отрисовке радара, линии и зеленых точек

    :parameters
    screen : pygame.display
        окно, в котором происходит отрисовка
    rect : image.get_rect
        фигура машины
    angle : int
        угол машины
    x : int
        координата x радара
    y : int
        координата y радара
    radar_angle : int
        угол радара
    length : int
        длина линии сканера

    :returns
    length : int
        новая длина линии сканера
    x : int
        новая координата x радара
    y : int
        новая координата y радара
    """
    while not screen.get_at((x, y)) == pygame.Color(2, 105, 31, 255) and length < 200:
        length += 1
        x = int(rect.center[0] + math.cos(math.radians(angle + radar_angle)) * length)
        y = int(rect.center[1] - math.sin(math.radians(angle + radar_angle)) * length)
    return length, x, y

```

Рисунок 3.1 – метод scanner класса DrawRadar

```

2 usages (1 dynamic)  yatim-dev
def draw(self, screen, rect, x, y):
    """Метод по отрисовке радара, линии и зеленых точек

    :parameters
    screen : pygame.display
        окно, в котором происходит отрисовка
    rect : image.get_rect
        фигура машины
    x : int
        координата x радара
    y : int
        координата y радара
    """
    pygame.draw.line(screen, (255, 255, 255, 255), rect.center, (x, y), 1)
    pygame.draw.circle(screen, (0, 255, 0, 0), (x, y), 3)

```

Рисунок 3.2 – метод draw класса DrawRadar

```

1 usage  yatim-dev
def draw_col_points(self, screen, collision_point_right, collision_point_left):
    """Метод по отрисовке точек коллизий

    :parameter
    screen : pygame.display
        окно, в котором происходит отрисовка
    collision_point_right : list[int]
        позиция левой точки коллизии
    collision_point_left : list[int]
        позиция левой точки коллизии
    """
    pygame.draw.circle(screen, (0, 255, 255, 0), collision_point_right, 4)
    pygame.draw.circle(screen, (0, 255, 255, 0), collision_point_left, 4)

```

Рисунок 3.3 – метод draw_col_points класса DrawRadar

4) start_window.py

Имеет одну функцию, которую необходимо рассмотреть

```

2 usages  yatim-dev
class StartWindow:
    """Класс по отрисовке начального окна с выбором карт"""

    yatim-dev
    def __init__(self, config_path):
        self.config_path = config_path
        # Инициализировать Pygame
        pygame.init()

        # Задать размеры окна
        window_width = 1244
        window_height = 1016

        # Создаем окно
        window = pygame.display.set_mode((window_width, window_height))
        pygame.display.set_caption("ai_drive")

        # Загрузка изображений
        map1 = pygame.image.load(os.path.join("Assets", "1.png"))
        map2 = pygame.image.load(os.path.join("Assets", "2.png"))
        map3 = pygame.image.load(os.path.join("Assets", "3.png"))
        logo = pygame.image.load(os.path.join("Assets", "logo.png"))

        # Установка исходного изображения на None
        current_map = None

        # Создание кнопок
        button_width = 200
        button_height = 100
        button_margin = 20

```

Рисунок 4.1.1 – инициализатор класса StartWindow

```

button_total_width = 3 * button_width + 2 * button_margin
button_start_x = (window_width - button_total_width) // 2
button_start_y = (window_height - button_height) // 2

button1_rect = pygame.Rect(
    button_start_x,
    button_start_y + 350,
    button_width,
    button_height,
)
button2_rect = pygame.Rect(
    button_start_x + button_width + button_margin,
    button_start_y + 350,
    button_width,
    button_height,
)
button3_rect = pygame.Rect(
    button_start_x + 2 * (button_width + button_margin),
    button_start_y + 350,
    button_width,
    button_height,
)

# Загрузка картинок для кнопок
button_image1 = pygame.image.load(os.path.join("Assets", "button_image1.png"))
button_image2 = pygame.image.load(os.path.join("Assets", "button_image2.png"))
button_image3 = pygame.image.load(os.path.join("Assets", "button_image3.png"))

```

Рисунок 4.1.2 – инициализатор класса StartWindow


```

# Высчитывание позиции logo
logo_rect = logo.get_rect(center=(window_width // 2, 250))

# Запустить игровой цикл
running = True
while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False
        elif event.type == MOUSEBUTTONDOWN:
            # Проверяем, была ли нажата кнопка
            if button1_rect.collidepoint(event.pos):
                current_map = map1
            elif button2_rect.collidepoint(event.pos):
                current_map = map2
            elif button3_rect.collidepoint(event.pos):
                current_map = map3

    # Очистить окно с серым фоном
    window.fill((200, 200, 200))

    # Отображать логотип вверху по центру
    window.blit(logo, logo_rect)

    # Рисуем кнопки
    window.blit(button_image1, button1_rect)
    window.blit(button_image2, button2_rect)
    window.blit(button_image3, button3_rect)

    # Если карта выбрана, то стартуем саму игру
    if current_map:
        GameWindow(window, current_map, self.config_path)

    # Обновить дисплей
    pygame.display.update()

# Выйти из Pygame
pygame.quit()

```

Рисунок 4.1.3 – инициализатор класса StartWindow

5) game_neat.py

Имеет 7 функций, которые необходимо рассмотреть

```

2 usages  yatim-dev
class GameWindow:
    """Класс для управления машиной алгоритмом NEAT"""

    yatim-dev
    def __init__(self, window, map, config_path):
        self.window = window
        self.map = map
        self.config_path = config_path
        self.setup_NEAT(config_path)

```

Рисунок 5.1 – инициализатор класса GameWindow

```

1 usage  yatim-dev *
def setup_NEAT(self, config_path):
    """Настройка NEAT"""
    config = neat.config.Config(
        neat.DefaultGenome,
        neat.DefaultReproduction,
        neat.DefaultSpeciesSet,
        neat.DefaultStagnation,
        config_path
    )

    global pop
    pop = neat.Population(config)

    pop.add_reporter(neat.StdOutReporter(True))

    pop.run(self.eval_genomes, 15)

```

Рисунок 5.2 – метод setup_NEAT класса GameWindow

```

1 usage  👤 yatim-dev
def eval_genomes(self, genomes, config):
    """Метод создания коллекций"""
    global cars, ge, nets

    cars = []
    ge = []
    nets = []

    # Заполняем списки
    for genome_id, genome in genomes:
        cars.append(dc.spawn_car(self.window))
        ge.append(genome)
        net = neat.nn.FeedForwardNetwork.create(genome, config)
        nets.append(net)
        # начальное значение
        genome.fitness = 0

    self.game()

```

Рисунок 5.3 – метод eval_genomes класса GameWindow

```

1 usage  👤 yatim-dev
def game(self):
    """Управление машиной и отрисовкой"""
    run = True
    while run:
        # Выйти по крестик
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # отрисовка карты
        self.window.blit(self.map, (0, 0))

        # конец поколения если все машины мертвы
        if len(cars) == 0:
            break

        self.car_handler()
        self.turn()

        dc.update_car(cars, self.window)

```

Рисунок 5.4 – метод game класса GameWindow

```

1 usage  👤 yatim-dev
def car_handler(self):
    """Обработчик fitness каждого автомобиля"""
    # Увеличиваем fitness каждого автомобиля, если тот еще жив
    for i, car in enumerate(cars):
        ge[i].fitness += 1
        if ge[i].fitness > 1000:
            print(ge[i])
            sys.exit()

        # Удаляем автомобиль, если он въехал в траву
        if not car.sprite.alive:
            self.remove(i)

```

Рисунок 5.5 – метод car_handler класса GameWindow

```

1 usage  👤 yatim-dev
def turn(self):
    """Определяем поведение(поворот) каждого автомобиля"""
    for i, car in enumerate(cars):
        # Получаем выходные данные нейронной сети
        output = nets[i].activate(car.sprite.data())
        # Поворот направо
        if output[0] > 0.7:
            car.sprite.direction = 1
        # Поворот налево
        if output[1] > 0.7:
            car.sprite.direction = -1
        # Не поворачиваем
        if output[0] <= 0.7 and output[1] <= 0.7:
            car.sprite.direction = 0

```

Рисунок 5.6 – метод turn класса GameWindow

```

1 usage  👤 yatim-dev
def remove(self, index):
    """Удаление автомобиля, коротый выехал на траву  

    :parameter  

    index : int  

    индекс автомобиля, коротый выехал на траву  

    """
    # удаляем машину  

    cars.pop(index)
    # удаляем генет  

    ge.pop(index)
    # удаляем нейронную сеть  

    nets.pop(index)

```

Рисунок 5.7 – метод remove класса GameWindow

3.2 Скриншоты игры

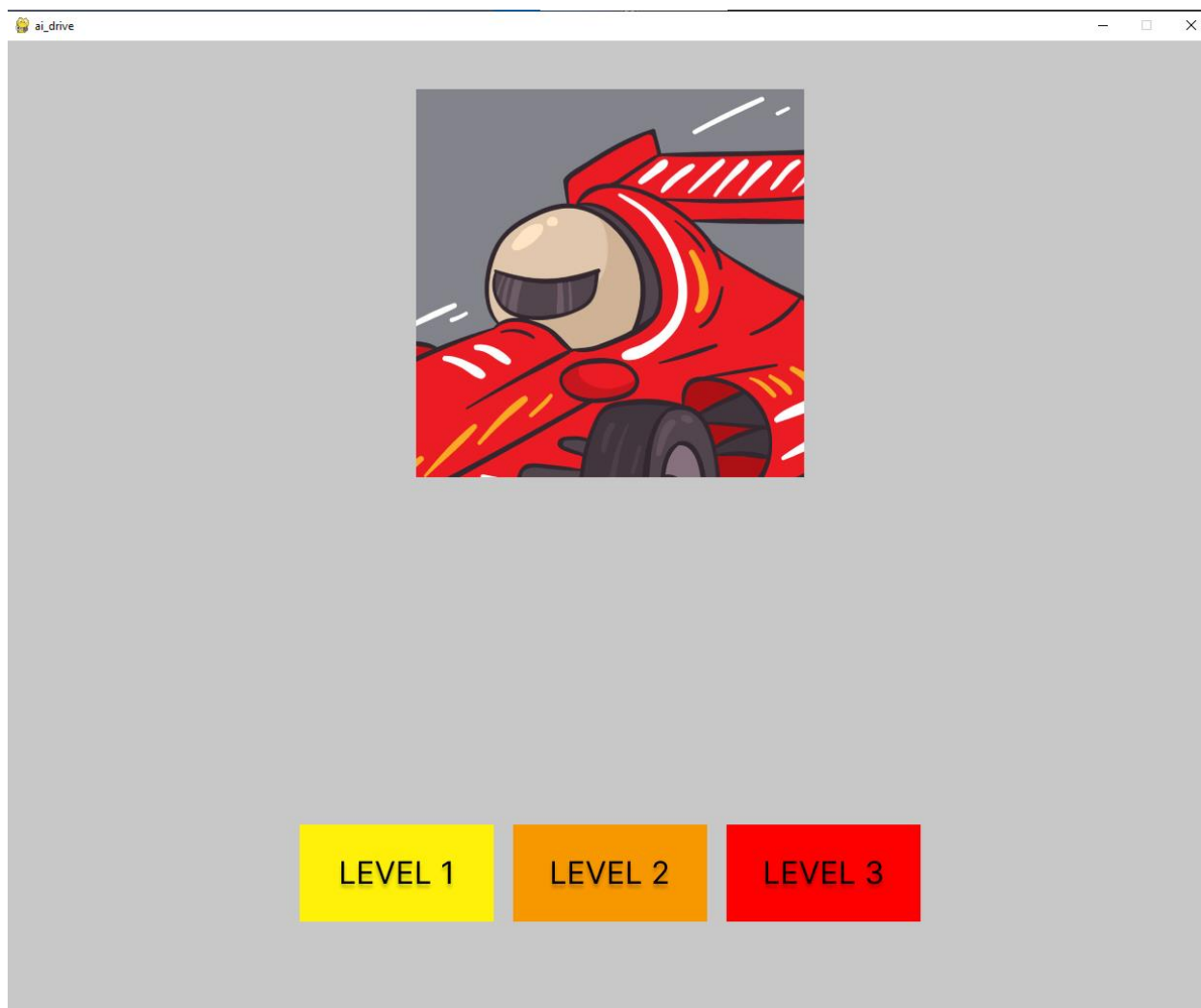


Рисунок 6.1 – Стартовое окно

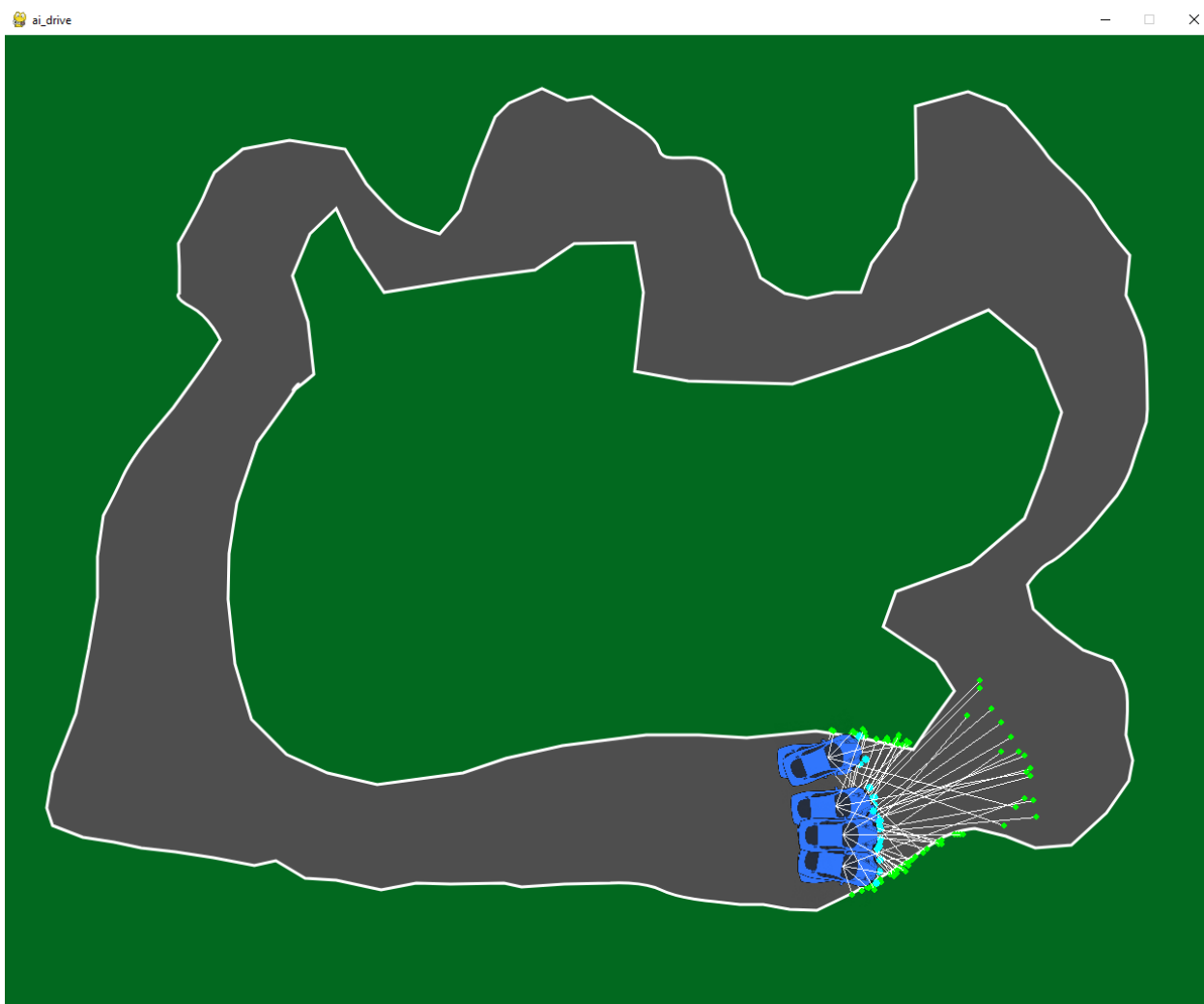


Рисунок 6.2 – Процесс игры

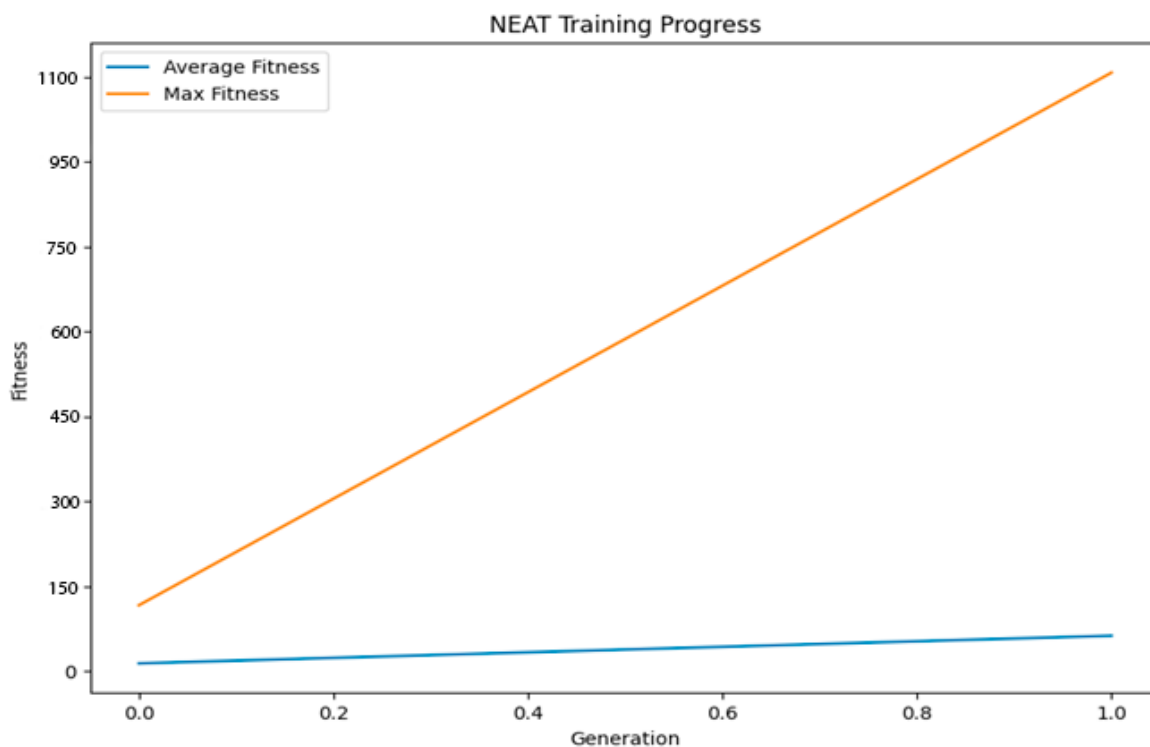


Рисунок 6.3.1 – итоги игры

```
Key: 129
Fitness: 1000
Nodes:
  0 DefaultNodeGene(key=0, bias=-0.8966293230948046, response=1.0, activation=tanh, aggregation=sum)
  1 DefaultNodeGene(key=1, bias=1.2881663919271216, response=1.0, activation=tanh, aggregation=sum)
Connections:
  DefaultConnectionGene(key=(-5, 0), weight=-0.2837518680985226, enabled=True)
  DefaultConnectionGene(key=(-5, 1), weight=1.6918354418308996, enabled=True)
  DefaultConnectionGene(key=(-4, 0), weight=-2.1129255701041956, enabled=False)
  DefaultConnectionGene(key=(-4, 1), weight=0.9227785978478937, enabled=True)
  DefaultConnectionGene(key=(-3, 0), weight=1.2456280308975922, enabled=True)|
  DefaultConnectionGene(key=(-3, 1), weight=-0.5336103562333567, enabled=True)
  DefaultConnectionGene(key=(-2, 0), weight=-0.005578356120388511, enabled=True)
  DefaultConnectionGene(key=(-2, 1), weight=-1.6683183860329263, enabled=True)
  DefaultConnectionGene(key=(-1, 1), weight=0.3431698481284965, enabled=True)
```

Рисунок 6.3.2 – итоги игры

На рисунке 6.3.2 выводится статистика и параметры победителя.

ЗАКЛЮЧЕНИЕ

В данной работе было исследовано применение библиотеки NEAT-Python для реализации эволюционного обучения нейросетей на примере игры. Главной целью работы было изучение принципов и возможностей

NEAT-Python, а также разработка программной реализации игры, использующей эволюционное обучение для управления игровым персонажем.

Изначально были изучены основные концепции нейросетей и глубокого обучения в теоретическом аспекте. Были рассмотрены принципы работы нейронных сетей, их структура и преимущества глубокого обучения при работе с большими объемами данных.

Далее была проведена детальная работа с библиотекой NEAT-Python. Была выполнена установка и изучены основные классы и методы, предоставляемые библиотекой. Это позволило успешно интегрировать NEAT-Python в разработку программной реализации игры.

Программная реализация игры включала создание игровой среды и интеграцию NEAT-Python для эволюции нейросетей. Были определены правила и механика игры, а также настроены параметры обучения. В процессе обучения происходила эволюция нейросетей, которые впоследствии могли управлять персонажем игры с использованием выработанных стратегий.

Анализ результатов обучения позволил сделать выводы о применимости NEAT-Python и эволюционного обучения в контексте игр. Была оценена производительность нейросетей, а также их способность адаптироваться и находить оптимальные стратегии для достижения поставленных целей в игровой среде.

В заключение, данная работа позволила более глубоко изучить принципы нейросетей, глубокого обучения и эволюционных алгоритмов с использованием NEAT-Python. Реализация игры с помощью NEAT-Python демонстрирует потенциал эволюционного обучения в создании интеллектуальных систем. Результаты исследования могут быть полезны для дальнейших исследований и разработок в области нейронных сетей и

игрового искусственного интеллекта. NEAT-Python представляет мощный инструмент для разработки и исследования алгоритмов эволюционного обучения и нейроэволюции в широком спектре приложений.

СПИСОК ЛИТЕРАТУРЫ

Ссылки на электронные ресурсы:

1 Курс Deep Learning (семестр 1, осень 2021): продвинутый поток URL:

<https://stepik.org/course/101721>

2 Efficient Evolution of Neural Network Topologies. Kenneth O. Stanley and Risto Miikkulainen. Department of Computer Sciences. The University of Texas at Austin. URL:

<https://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf>

3 Документация по модулю NEAT-Python. URL:

<https://neat-python.readthedocs.io/en/latest/index.html>

4 Я. Гудфеллоу, И. Бенджио, А. Курвилль. Глубокое обучение. URL:

https://library.kre.dp.ua/Books/2-4%20kurs/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%20+%20%D0%BC%D0%BE%D0%B2%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9%20%D1%96%D0%BD%D1%82%D0%B5%D0%B%D0%B5%D0%BA%D1%82/Machineobuchenie@bzd_channel.pdf

ПРИЛОЖЕНИЯ

Приложение 1. main.py

```
1  import os
2
3  from views.start_window import StartWindow
4
5  if __name__ == "__main__":
6      local_dir = os.path.dirname(__file__)
7      config_path = os.path.join(local_dir, 'config.txt')
8      StartWindow(config_path)
```

Приложение 2. config.txt

```
1 [NEAT]
2 fitness_criterion      = max
3 fitness_threshold      = 10000
4 pop_size               = 50
5 reset_on_extinction    = True
6
7 [DefaultGenome]
8 ▶ activation_default    = tanh
9 activation_mutate_rate  = 0.0
10 activation_options     = tanh
11
12 ▶ aggregation_default   = sum
13 aggregation_mutate_rate = 0.0
14 aggregation_options    = sum
15
16 # node bias options
17 bias_init_mean          = 0.0
18 bias_init_stdev         = 1.0
19 bias_max_value          = 30.0
20 bias_min_value          = -30.0
21 bias_mutate_power       = 0.5
22 bias_mutate_rate        = 0.7
23 bias_replace_rate       = 0.1
24
25 ▶ compatibility_disjoint_coefficient = 1.0
26 compatibility_weight_coefficient    = 0.5
27
28 ▶ conn_add_prob          = 0.5
29 conn_delete_prob        = 0.5
30
31 ▶ enabled_default        = True
32 enabled_mutate_rate      = 0.01
33
34 feed_forward             = True
35 initial_connection       = full
36
37 ▶ node_add_prob          = 0.2
38 node_delete_prob         = 0.2
```

```

39
40     ▶ num_hidden          = 0
41     num_inputs          = 5
42     num_outputs         = 2
43
44     ▶ response_init_mean  = 1.0
45     response_init_stdev  = 0.0
46     response_max_value   = 30.0
47     response_min_value   = -30.0
48     response_mutate_power = 0.0
49     response_mutate_rate  = 0.0
50     response_replace_rate = 0.0
51
52     ▶ weight_init_mean   = 0.0
53     weight_init_stdev   = 1.0
54     weight_max_value     = 30
55     weight_min_value     = -30
56     weight_mutate_power  = 0.5
57     weight_mutate_rate   = 0.8
58     weight_replace_rate  = 0.1
59
60     [DefaultSpeciesSet]
61     compatibility_threshold = 3.0
62
63     [DefaultStagnation]
64     species_fitness_func = max
65     max_stagnation       = 20
66     species_elitism       = 2
67
68     [DefaultReproduction]
69     elitism               = 2
70     survival_threshold    = 0.2

```