

Разработка и обучение нейросети для автономного управления в игре Flappy Bird с использованием алгоритма NEAT

Акентьев Данила Денисович

*Приамурский государственный университет имени Шолом-Алейхема
студент*

Аннотация

В данной работе рассматривается создание модели искусственного интеллекта на основе нейронных сетей, способной самостоятельно обучаться и успешно управлять персонажем в игре Flappy Bird. Используется алгоритм эволюционного обучения NEAT (NeuroEvolution of Augmenting Topologies), позволяющий одновременно оптимизировать структуру и веса нейросети. В результате реализован прототип, в котором множество виртуальных агентов (птиц) обучаются преодолевать препятствия, улучшая свои навыки от поколения к поколению без вмешательства человека.

Ключевые слова: Flappy Bird, искусственный интеллект, нейронные сети, NEAT, эволюционное обучение, автономное управление, самообучающаяся система.

Development and training of a neural network for autonomous control in the Flappy Bird game using the NEAT algorithm

Akentev Danila Denisovich

*Sholom-Aleichem Priamursky State University
Student*

Abstract

This paper considers the creation of an artificial intelligence model based on neural networks capable of self-learning and successfully controlling a character in the Flappy Bird game. The NEAT (NeuroEvolution of Augmenting Topologies) evolutionary learning algorithm is used, which simultaneously optimizes the structure and weights of the neural network. As a result, a prototype has been implemented in which many virtual agents (birds) are trained to overcome obstacles, improving their skills from generation to generation without human intervention.

Keywords: Flappy Bird, artificial intelligence, neural networks, NEAT, evolutionary learning, autonomous control, self-learning system.

1 Введение

1.1 Актуальность

С развитием искусственного интеллекта и машинного обучения возникает всё большая потребность в разработке автономных систем, способных обучаться и адаптироваться к сложным условиям без постоянного

контроля человека. Игры, в частности простые аркады, такие как Flappy Bird, предоставляют удобную тестовую среду для исследований алгоритмов обучения агентов и оптимизации поведения с использованием нейронных сетей и эволюционных методов. Использование алгоритма NEAT позволяет не только обучать весовые параметры, но и оптимизировать архитектуру сети, что существенно расширяет возможности создания эффективных моделей ИИ. Это делает разработку и исследование подобных систем актуальной задачей для современного направления искусственного интеллекта и робототехники.

1.2 Обзор исследований

В работе Pătraşcu C.B. и Iancu D.T. рассмотрено применение алгоритма NEAT (NeuroEvolution of Augmenting Topologies) для обучения агентов в простых играх, таких как Dinosaur Game и Flappy Bird. Авторы показали, что метод нейроэволюции позволяет эффективно обучать игровые агенты без необходимости специализированных вычислительных устройств, что расширяет возможности использования данного подхода в широком спектре игровых приложений [1].

Статья Piyush N.S., Yash J.C. и Shubham G.C. посвящена процессу создания игры на языке Python с использованием библиотеки Pygame. Авторы последовательно описывают этапы разработки игры Dracula — от создания графического интерфейса до внедрения игровых элементов и уровней. Работа демонстрирует возможности и ограничения Pygame как инструмента для разработки 2D-игр, что важно для начинающих разработчиков [2].

В исследовании В.И. Олексюка рассмотрены специфика и возможности применения методов машинного обучения в компьютерных играх. Автор подчёркивает, что использование глубокого обучения в игровых средах позволяет улучшать качество игрового процесса и снижать риски при обучении моделей, что делает данный подход актуальным для современной игровой индустрии [3].

Работа Д.Д. Абросимова, Р.В. Кононова и М.Н. Беленькой представляет обзор алгоритмов машинного обучения и их применения в игровых проектах. Особое внимание уделено преимуществам внедрения ML-технологий для повышения адаптивности и сложности игровых систем, на примере игры жанра «гонки». Авторы подчёркивают перспективность машинного обучения для создания интеллектуальных игровых агентов [4].

В статье И.И. Чернявского рассматривается задача автоматического построения управляющих автоматов на примере игры Robocode с использованием методов машинного обучения. Предложенный подход сравнивается с генетическим программированием, что позволяет выявить преимущества и эффективность применения ML для автоматизации создания стратегий управления игровыми агентами [5].

1.3 Цель исследования

Целью данного исследования является разработка и реализация автономного агента для игры Flappy Bird на основе нейросетевого подхода с

использованием алгоритма NEAT, который способен самостоятельно обучаться эффективно преодолевать игровые препятствия и достигать максимального результата без внешнего вмешательства.

2 Материалы и методы

В качестве игровой среды использовалась популярная аркадная игра Flappy Bird, реализованная с использованием библиотеки Pygame на языке Python. Для обучения автономного агента применён алгоритм NEAT (NeuroEvolution of Augmenting Topologies), реализованный через пакет neat-python, который позволяет эволюционно оптимизировать как структуру, так и веса нейросети.

Обучающаяся система состоит из множества агентов (птиц), каждый из которых управляется собственной нейросетью. Каждый агент получает на вход параметры текущего состояния игры, такие как вертикальная позиция птицы и расстояние до ближайших препятствий, и принимает решение — прыгать или нет. Фитнес-функция алгоритма оценивает успешность агента по количеству пройденных препятствий и выживанию.

Эволюция происходит по стандартному циклу: каждый агент оценивается в игровом процессе, затем на основе результатов формируется новое поколение с вариациями в архитектуре и параметрах сетей. Для визуализации и отладки использовался графический интерфейс Pygame.

3 Результаты и обсуждения

1. Импорты и инициализация (рис.1).

A screenshot of a code editor with a dark background. The code is written in a syntax-highlighted font. It shows several import statements: 'import pygame' (green), 'import neat' (green), 'import time' (green), 'import os' (green), and 'import random' (green). Below these, there is a call to 'pygame.font.init()' (green).

Рисунок 1. Код

Импортируются необходимые библиотеки:

- pygame — для графики и управления игрой.
- neat — для реализации алгоритма нейронной эволюции NEAT.
- time, os, random — системные модули для времени, работы с файлами и случайных чисел.

Инициализация шрифтов в Pygame (pygame.font.init()).

2. Глобальные константы и загрузка ресурсов (рис.2).

```

WIN_WIDTH = 500
WIN_HEIGHT = 800

BIRD_IMGS = [
    pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "yellowbird-upflap.png"))),
    pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "yellowbird-midflap.png"))),
    pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "yellowbird-downflap.png")))
]

PIPE_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "pipe-green.png")))
BASE_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "base.png")))
BG_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs", "background-day.png")))

STAT_FONT = pygame.font.SysFont("comicsans", 50)

```

Рисунок 2. Код

- Определены размеры окна игры.
- Загружаются и масштабируются изображения птиц, труб, базы и фона.
- Создаётся шрифт для отображения счёта и текущего поколения.

3. Класс Bird — описание птицы (рис.3,4).

```

class Bird:
    IMGS = BIRD_IMGS
    MAX_ROTATION = 25
    ROT_VEL = 20
    ANIMATION_TIME = 5

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.tilt = 0
        self.tick_count = 0
        self.vel = 0
        self.height = self.y
        self.img_count = 0
        self.img = self.IMGS[0]

    def jump(self):
        self.vel = -10.5
        self.tick_count = 0
        self.height = self.y

    def move(self):
        self.tick_count += 1
        d = self.vel * self.tick_count + 1.5 * self.tick_count ** 2

        if d >= 16:
            d = 16
        if d < 0:
            d -= 2

        self.y = self.y + d

        if d < 0 or self.y < self.height + 50:
            if self.tilt < self.MAX_ROTATION:
                self.tilt = self.MAX_ROTATION
        else:
            if self.tilt > -90:
                self.tilt -= self.ROT_VEL

```

Рисунок 3. Код

```
def draw(self, win):
    self.img_count += 1

    if self.img_count < self.ANIMATION_TIME:
        self.img = self.IMGS[0]
    elif self.img_count < self.ANIMATION_TIME*2:
        self.img = self.IMGS[1]
    elif self.img_count < self.ANIMATION_TIME*3:
        self.img = self.IMGS[2]
    elif self.img_count < self.ANIMATION_TIME*4:
        self.img = self.IMGS[1]
    elif self.img_count == self.ANIMATION_TIME*4 + 1:
        self.img = self.IMGS[0]
        self.img_count = 0

    if self.tilt <= -80:
        self.img = self.IMGS[1]
        self.img_count = self.ANIMATION_TIME*2

    rotated_image = pygame.transform.rotate(self.img, self.tilt)
    new_rect = rotated_image.get_rect(center=self.img.get_rect(topleft=(self.x, self.y)).center)

    win.blit(rotated_image, new_rect.topleft)

def get_mask(self):
    return pygame.mask.from_surface(self.img)
```

Рисунок 4. Код

Поля: изображения анимации, угол наклона, скорость, позиция.

Методы:

- `jump()` — задаёт отрицательную скорость для прыжка.
- `move()` — рассчитывает движение птицы с гравитацией, меняет угол наклона.
- `draw(win)` — рисует птицу на экране с анимацией и поворотом.
- `get_mask()` — возвращает маску изображения для точного определения столкновений.

4. Класс `Pipe` — описание труб (рис.5).

```
class Pipe:
    GAP = 200
    VEL = 5

    def __init__(self, x):
        self.x = x
        self.height = 0
        self.gap = 100
        self.top = 0
        self.bottom = 0
        self.PIPE_TOP = pygame.transform.flip(PIPE_IMG, False, True)
        self.PIPE_BOTTOM = PIPE_IMG
        self.passed = False
        self.set_height()

    def set_height(self):
        self.height = random.randrange(50, 450)
        self.top = self.height - self.PIPE_TOP.get_height()
        self.bottom = self.height + self.GAP

    def move(self):
        self.x -= self.VEL

    def draw(self, win):
        win.blit(self.PIPE_TOP, (self.x, self.top))
        win.blit(self.PIPE_BOTTOM, (self.x, self.bottom))

    def collide(self, bird):
        bird_mask = bird.get_mask()
        top_mask = pygame.mask.from_surface(self.PIPE_TOP)
        bottom_mask = pygame.mask.from_surface(self.PIPE_BOTTOM)

        top_offset = (self.x - bird.x, self.top - round(bird.y))
        bottom_offset = (self.x - bird.x, self.bottom - round(bird.y))

        b_point = bird_mask.overlap(bottom_mask, bottom_offset)
        t_point = bird_mask.overlap(top_mask, top_offset)

        return t_point or b_point
```

Рисунок 5. Код

Поля: позиция, высота, верхнее и нижнее изображение трубы, состояние прохождения.

Методы:

- `set_height()` — случайно устанавливает высоту трубы.
- `move()` — сдвигает трубу влево.
- `draw(win)` — рисует верхнюю и нижнюю части трубы.
- `collide(bird)` — проверяет столкновение трубы с птицей с помощью масок.

5. Класс `Base` — описание движущейся земли (рис.6).

```

class Base:
    VEL = 5
    WIDTH = BASE_IMG.get_width()
    IMG = BASE_IMG

    def __init__(self, y):
        self.y = y
        self.x1 = 0
        self.x2 = self.WIDTH

    def move(self):
        self.x1 -= self.VEL
        self.x2 -= self.VEL

        if self.x1 + self.WIDTH < 0:
            self.x1 = self.x2 + self.WIDTH

        if self.x2 + self.WIDTH < 0:
            self.x2 = self.x1 + self.WIDTH

    def draw(self, win):
        win.blit(self.IMG, (self.x1, self.y))
        win.blit(self.IMG, (self.x2, self.y))

```

Рисунок 6. Код

Поля: скорость движения, ширина изображения, позиции двух частей базы.

Методы:

- `move()` — двигает обе части базы влево, закливая их.
- `draw(win)` — отрисовывает базу.

6. Функция `draw_window` — отрисовка экрана (рис.7).

```

def draw_window(win, birds, pipes, base, score, gen):
    win.blit(BG_IMG, (0, 0))

    for pipe in pipes:
        pipe.draw(win)

    text = STAT_FONT.render("Score: " + str(score), 1, (255, 255, 255))
    win.blit(text, (WIN_WIDTH - 10 - text.get_width(), 10))

    text = STAT_FONT.render("Gen: " + str(gen), 1, (255, 255, 255))
    win.blit(text, (10, 10))

    base.draw(win)

    for bird in birds:
        bird.draw(win)

    pygame.display.update()

```

Рисунок 7. Код

Отрисовывает фон, трубы, базу, счёт, номер поколения и птиц. Обновляет экран игры (`pygame.display.update()`).

7. Функция `main` — основной игровой цикл и логика NEAT (рис.8,9).

```

def main(genomes, config):
    global gen
    gen += 1

    nets = []
    ge = []
    birds = []

    for genome_id, genome in genomes:
        genome.fitness = 0
        net = neat.nn.FeedForwardNetwork.create(genome, config)
        nets.append(net)
        birds.append(Bird(230, 350))
        ge.append(genome)

    base = Base(730)
    pipes = [Pipe(600)]
    win = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
    clock = pygame.time.Clock()
    score = 0

    run = True
    while run and len(birds) > 0:
        clock.tick(30)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.quit()
                quit()

        pipe_ind = 0
        if len(pipes) > 1 and birds[0].x > pipes[0].x + pipes[0].PIPE_TOP.get_width():
            pipe_ind = 1

        for x, bird in enumerate(birds):
            bird.move()
            ge[x].fitness += 0.1

            output = nets[x].activate((bird.y, abs(bird.y - pipes[pipe_ind].height), abs(bird.y - pipes[pipe_ind].bottom)))

            if output[0] > 0.5:
                bird.jump()

```

Рисунок 8. Код

```

    rem = []
    add_pipe = False
    for pipe in pipes:
        for x, bird in enumerate(birds):
            if pipe.collide(bird):
                ge[x].fitness -= 1
                birds.pop(x)
                nets.pop(x)
                ge.pop(x)

            if not pipe.passed and birds and pipe.x < birds[0].x:
                pipe.passed = True
                add_pipe = True

        pipe.move()

    if add_pipe:
        score += 1
        for genome in ge:
            genome.fitness += 5
        pipes.append(Pipe(600))

    for pipe in pipes:
        if pipe.x + pipe.PIPE_TOP.get_width() < 0:
            rem.append(pipe)

    for r in rem:
        pipes.remove(r)

    for x, bird in enumerate(birds):
        if bird.y + bird.img.get_height() >= 730 or bird.y < 0:
            birds.pop(x)
            nets.pop(x)
            ge.pop(x)

    base.move()
    draw_window(win, birds, pipes, base, score, gen)

```

Рисунок 9. Код

Получает текущие геномы и конфигурацию NEAT. Создаёт нейросети (nets), птиц и устанавливает фитнес каждому геному. Инициализирует базу и трубы, создаёт окно и запускает цикл игры.

В цикле:

- Обрабатываются события (выход из игры).
- Определяется ближайшая труба для каждой птицы (pipe_ind).
- Для каждой птицы:
 - Вызывается move().
 - Увеличивается фитнес за выживание.
 - Получаются входы для нейросети: высота птицы, расстояния до верхней и нижней трубы.
 - Нейросеть принимает решение, прыгать или нет.
- Проверяется столкновение птиц с трубами, птицы, столкнувшиеся, удаляются и их фитнес уменьшается.
- Добавляется новая труба, если птицы успешно прошли через предыдущую.
- Удаляются трубы, вышедшие за экран.
- Проверяется, не вышли ли птицы за границы экрана (вниз или вверх).
- Двигается база и вызывается draw_window для обновления экрана.

8. Функция run — запуск NEAT (рис.10).

```
def run(config_file):  
    config = neat.config.Config(  
        neat.DefaultGenome,  
        neat.DefaultReproduction,  
        neat.DefaultSpeciesSet,  
        neat.DefaultStagnation,  
        config_file,  
    )  
  
    p = neat.Population(config)  
  
    p.add_reporter(neat.StdOutReporter(True))  
    p.add_reporter(neat.StatisticsReporter())  
  
    p.run(main, 50)
```

Рисунок 10. Код

Загружает конфигурацию NEAT из файла. Создаёт популяцию. Добавляет отчёты о ходе обучения. Запускает эволюцию, вызывая main в качестве фитнес-функции, на 50 поколений.

9. Запуск программы (рис.11).

```
if __name__ == "__main__":  
    local_dir = os.path.dirname(__file__)  
    config_path = os.path.join(local_dir, "config-feedforward.txt")  
    run(config_path)
```

Рисунок 11. Код

Определяет путь к файлу конфигурации. Запускает обучение с помощью run().

10. Внешний вид игры (рис.12).

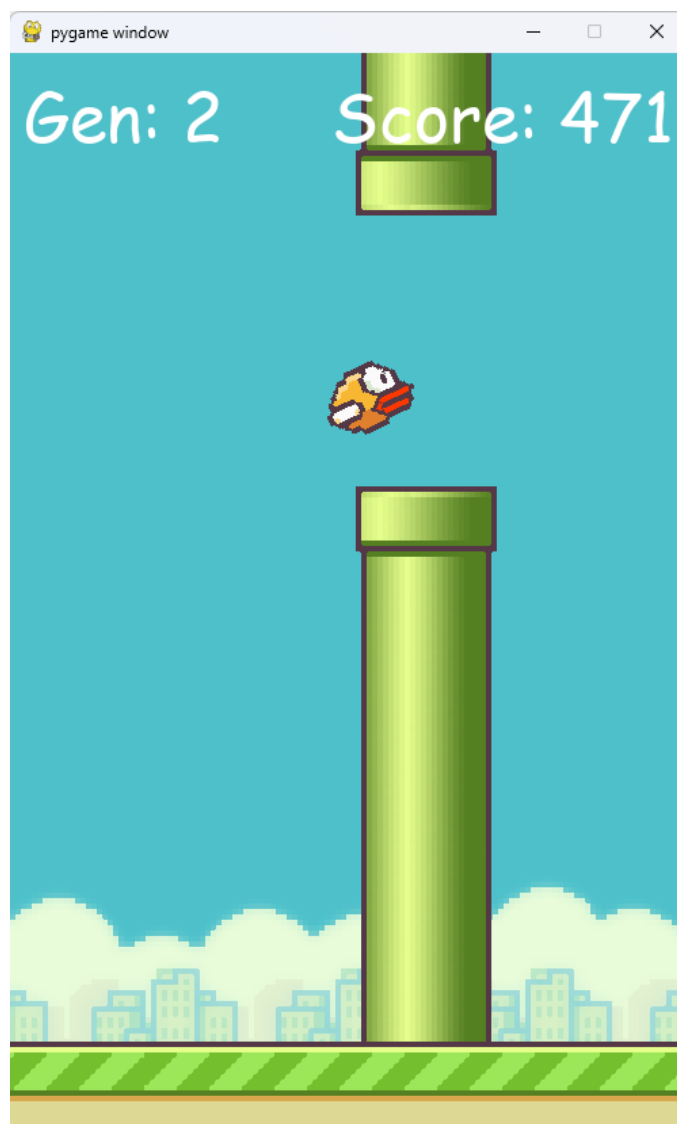


Рисунок 12. Игра

11. Конфиг для игры (рис.13,14).

```
[NEAT]
fitness_criterion      = max
fitness_threshold      = 1000
pop_size               = 50
reset_on_extinction    = False

[DefaultGenome]

feed_forward = True

response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power    = 0.0
response_mutate_rate     = 0.0
response_replace_rate    = 0.0

# node activation options
activation_default      = sigmoid
activation_mutate_rate  = 0.0
activation_options      = sigmoid

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient  = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.3

# connection weight options
enabled_default         = True
enabled_mutate_rate     = 0.01

initial_connection      = full_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 0
num_inputs              = 3
num_outputs              = 1
```

Рисунок 13. Конфиг

```
# input/output nodes cannot be deleted or modified
input_nodes      = 3
output_nodes     = 1

# connection weight options
weight_init_mean  = 0.0
weight_init_stdev = 1.0
weight_max_value  = 30
weight_min_value  = -30
weight_mutate_power = 0.5
weight_mutate_rate = 0.8
weight_replace_rate = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 15
species_elitism       = 2

[DefaultReproduction]
elitism              = 2
survival_threshold   = 0.2
```

Рисунок 14. Конфиг

4 Выводы

В ходе выполнения проекта была реализована игра Flappy Bird с использованием библиотеки Pygame и алгоритма нейроэволюции NEAT для обучения искусственного интеллекта управлять птицами. Благодаря применению алгоритма NEAT птицы смогли самостоятельно обучаться преодолевать препятствия в виде труб без внешнего вмешательства.

Реализация включала создание классов для основных игровых объектов — птицы, труб и базы — с их визуализацией и логикой взаимодействия. Нейросети получили на вход информацию о текущем положении птицы и ближайших препятствиях, что позволило им принимать решения о прыжках. В процессе эволюции, при постепенном улучшении популяции, наблюдалось устойчивое повышение производительности и выживаемости птиц.

Таким образом, проект демонстрирует эффективность нейроэволюционных методов в задачах обучения с подкреплением в игровых симуляциях. Работа также показывает, как можно интегрировать классические библиотеки для разработки игр с современными алгоритмами искусственного интеллекта, что открывает широкие возможности для дальнейших исследований и применения в области игр и автономных систем.

Библиографический список

1. Pătrașcu C.B. и Iancu D.T. Neat algorithm for simple games // 2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). 2023. С. 1-6.
2. Piyush N.S., Yash J.C., Shubham G.C. Pygame: Develop Games using Python
3. // International Journal for Research in Applied Science & Engineering Technology (IJRASET). 2021. Т. 9. №. 6. С. 444-449.

4. Олексюк В.И. Возможности использования машинного обучение в играх // Академическая публицистика. 2021. №. 3. С. 123-127.
5. Абросимов Д.Д., Кононов Р.В., Беленькая М.Н. Машинное обучение в игровых проектах // Телекоммуникации и информационные технологии. 2019. Т. 6. №. 2. С. 146-149.
6. Чернявский И.И. Применение машинного обучения для создания управляющих автоматов на примере игры «Robocode» // Научно-технический вестник информационных технологий, механики и оптики. 2011. №. 2(72). С. 22-26.