**Жасанды нейронды желі алгоритмі бойынша құрылған T-Rex**

Біз жасайтын Жасанды нейрондар желі алгоритмі бойынша құрастырылған ойын идеясы-өз ойындарымызды бақылай отырып ойнауды үйрену. Жасанды нейрондар кез келген уақытта шешім қабылдауға мүмкіндік беретін интеллектке ие болады (нейрондық желі алгоритмі арқылы) және біз оның интеллектісі менің шешімдеріме мүмкіндігінше жақын болуы үшін алгоритмді қолданамыз.

Бұл келесі кедергіге байланысты. Кедергі болған кезде секіруге тура келеді, бірақ кедергіні жеңгеннен кейін міндетті түрде шегіну үшін секірмес бұрын күтуге тура келеді. Олардың бірнеше түрі бар: біреу ұзын, біреу ұзын, ал біреу құс (сондықтан биіктікте). Нақты айтқанда, біздің шешімдеріміз мыналарға байланысты болады деп қорытынды жасауға болады:

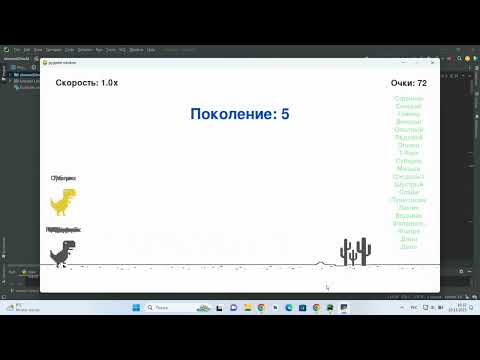
* Келесі кедергінің ені
* Келесі кедергінің ұзындығы
* Келесі кедергінің биіктігі
* Кактус пен келесі кедергі арасындағы қашықтық

Тағы бір ескеретін параметр бар: ойын алға жылжыған сайын біздің T-Rex жылдамырақ және жылдамырақ қозғалады. Жылдам қозғалғанда, ертерек секіруге тура келеді. Сондықтан біздің шешімдеріміз де жылдамдығымызға байланысты болады.

Міне, біздің жасанды нейрондарымыз неден тұрады: кез келген уақытта біз оған 5 параметрімізді береміз және ол бізге секіру керек пе, жоқ па, соны айтады. Біз бірнеше миллисекундтан кейін және ойынның соңына дейін қайтадан бастаймыз.

Бақылау: мен елестеткен нәрседе екі мүмкін мән қайтарады: біз секіреміз немесе секірмейміз. Бірақ екеуін де бір уақытта жасау мүмкін емес (немесе оның кез келгені). Сондықтан біз қорытындыны біреуіне дейін жеңілдетеміз: егер секірмесек 0-ге тең сан немесе секірсек 1-ге тең.

**Ойынның видеосы осы жерде (ютубқа жүктелген):**

**[](https://www.youtube.com/embed/I_Mdesvccos?feature=oembed)**

**Сілтеме:** [**https://youtu.be/I\_Mdesvccos?si=kfIzQEVjcELu-p00**](https://youtu.be/I_Mdesvccos?si=kfIzQEVjcELu-p00)

**Ойынның коды:**

import pygame

import random

import sys

import math

from enum import Enum

import neat

width = 1280

height = 720

bg = (255, 255, 255, 255)

score = 0

score\_speedup = 100

game\_speed = 8.0

skins = ["default", "aqua", "black", "bloody", "cobalt", "gold", "insta",

         "lime", "magenta", "magma", "navy", "neon", "orange", "pinky",

         "purple", "rgb", "silver", "subaru", "sunny", "toxic"]

names = ["Флафи", "Фалафель", "Ведьмак", "Лютик", "Пучеглазик", "Слайм", "Шустрый", "Следопыт",

         "Малыш", "Субарик", "Т-Рекс", "Птенец", "Рядовой", "Опытный", "Ветеран", "Геймер",

         "Самурай", "Странник"]

generation = 0

class DinoState(Enum):

    RUN = 1

    JUMP = 2

class Dino:

    name = "Carl"

    jump\_power = 10

    cur\_jump\_power = jump\_power

    color = "default"

    sprites = {

        "run": [],

        "jump": []

    }

    image = None

    run\_animation\_index = [0, 5]

    hitbox = None

    state = DinoState.RUN

    def \_\_init\_\_(self, x, y, color="default", name=None):

        self.color = color

        self.load\_sprites()

        self.hitbox = pygame.Rect(x, y, self.sprites["run"][0].get\_width(), self.sprites["run"][0].get\_height())

        self.image = self.sprites["run"][0]

        if name is not None:

            self.name = name

    def load\_sprites(self):

        self.sprites["jump"].append(pygame.image.load(f"sprites/dino/{self.color}\_jump.png"))

        self.sprites["run"].append(pygame.image.load(f"sprites/dino/{self.color}\_run1.png"))

        self.sprites["run"].append(pygame.image.load(f"sprites/dino/{self.color}\_run2.png"))

    def update(self):

        if self.state == DinoState.RUN:

            self.run()

        elif self.state == DinoState.JUMP:

            self.jump()

    def run(self):

        self.sprites["run"][0] = pygame.image.load(f"sprites/dino/{self.color}\_run1.png")

        self.sprites["run"][1] = pygame.image.load(f"sprites/dino/{self.color}\_run2.png")

        self.image = self.sprites["run"][self.run\_animation\_index[0] // self.run\_animation\_index[1]]

        self.run\_animation\_index[0] += 1

        if self.run\_animation\_index[0] >= self.run\_animation\_index[1] \* 2:

            self.run\_animation\_index[0] = 0

    def jump(self):

        if self.state == DinoState.JUMP:

            self.hitbox.y -= self.cur\_jump\_power \* (2 \* (game\_speed / 8))

            self.cur\_jump\_power -= 0.5 \* (game\_speed / 8)

            # if self.cur\_jump\_power <= -self.jump\_power:

            #    self.hitbox.y -= self.cur\_jump\_power \* (2 \* (game\_speed / 8))

            #    self.state = DinoState.RUN

            #    self.cur\_jump\_power = self.jump\_power

            if self.hitbox.y >= height - 170:

                self.hitbox.y = height - 170

                self.state = DinoState.RUN

                self.cur\_jump\_power = self.jump\_power

        else:

            self.state = DinoState.JUMP

            self.image = pygame.image.load(f"sprites/dino/{self.color}\_jump.png")

            # self.image = self.sprites["jump"][0]

    def draw(self, scr, fnt=None):

        scr.blit(self.image, (self.hitbox.x, self.hitbox.y))

        if fnt is not None:

            c\_label = fnt.render(self.name.capitalize(), True, (100, 100, 100))

            c\_label\_rect = c\_label.get\_rect()

            c\_label\_rect.center = (self.hitbox.x + 45, self.hitbox.y - 30)

            scr.blit(c\_label, c\_label\_rect)

class Cactus:

    available\_types = ["1", "2", "3", "4", "5", "6"]

    cactus\_type = None

    image = None

    hitbox = None

    is\_active = True

    def \_\_init\_\_(self, x, y, forced\_type=None):

        if forced\_type is not None:

            self.cactus\_type = forced\_type

        self.load\_image()

        self.hitbox.x = x

        self.hitbox.y = y - self.hitbox.height  # origin from bottom

    def randomize\_cactus(self):

        self.cactus\_type = random.choice(self.available\_types)

    def load\_image(self):

        if self.cactus\_type is None:

            self.randomize\_cactus()

        self.image = pygame.image.load(f"sprites/cactus/{self.cactus\_type}.png")

        self.hitbox = self.image.get\_rect()

    def update(self):

        self.hitbox.x -= game\_speed

        if self.hitbox.x < -self.hitbox.width:

            # remove this cactus

            self.is\_active = False

    def draw(self, scr):

        scr.blit(self.image, self.hitbox)

def calc\_dist(a, b):

    dx = a[0] - b[0]

    dy = a[1] - b[1]

    return math.sqrt(dx \*\* 2 + dy \*\* 2)

def run\_game(genomes, config):

    global game\_speed, score, enemies, dinosaurs, generation, score\_speedup

    generation += 1

    game\_speed = 8.0

    score = 0

    score\_speedup = 100

    enemies = [Cactus(width + 300 / random.uniform(0.8, 3), height - 85),

               Cactus(width \* 2 + 200 / random.uniform(0.8, 3), height - 85),

               Cactus(width \* 3 + 400 / random.uniform(0.8, 3), height - 85)]

    dinosaurs = []

    nets = []

    skins\_copy = skins[:]

    names\_copy = names[:]

    # init genomes

    for i, g in genomes:

        net = neat.nn.FeedForwardNetwork.create(g, config)

        nets.append(net)

        g.fitness = 0  # every genome is not successful at the start

        skin = "default"

        if len(skins\_copy):

            skin = skins\_copy.pop()

        name = "Дино"

        if len(names\_copy):

            name = names\_copy.pop()

        dinosaurs.append(Dino(30, height - 170, skin, name))

    # init

    pygame.init()

    screen = pygame.display.set\_mode((width, height))

    clock = pygame.time.Clock()

    road\_chunks = [

        [pygame.image.load('sprites/road.png'), [0, height - 100]],

        [pygame.image.load('sprites/road.png'), [2404, height - 100]]

    ]

    font = pygame.font.SysFont("Roboto Condensed", 30)

    score\_font = pygame.font.SysFont("Roboto Condensed", 40)

    dname\_font = pygame.font.SysFont("Roboto Condensed", 30)

    heading\_font = pygame.font.SysFont("Roboto Condensed", 70)

    # dinosaurs = [Dino(30, height-170, "subaru", "Howdy")]

    # the loop

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

        # display bg & road

        screen.fill(bg)

        for road\_chunk in road\_chunks:

            if road\_chunk[1][0] <= -2400:

                road\_chunk[1][0] = road\_chunks[len(road\_chunks) - 1][1][0] + 2400

                road\_chunks[0], road\_chunks[1] = road\_chunks[1], road\_chunks[0]

                break

            road\_chunk[1][0] -= game\_speed

            screen.blit(road\_chunk[0], (road\_chunk[1][0], road\_chunk[1][1]))

        # draw dino

        for dino in dinosaurs:

            dino.update()

            dino.draw(screen, font)

        # quit if there is no dinos left

        if len(dinosaurs) == 0:

            break

        # generate enemies

        if len(enemies) < 3:

            enemies.append(Cactus(enemies[len(enemies) - 1].hitbox.x + width / random.uniform(0.8, 3), height - 85))

        # draw enemies

        rem\_list = []

        for i, enemy in enumerate(enemies):

            enemy.update()

            enemy.draw(screen)

            if not enemy.is\_active:

                rem\_list.append(i)

                continue

            for j, dinosaur in enumerate(dinosaurs):

                if dinosaur.hitbox.colliderect(enemy.hitbox):

                    genomes[j][1].fitness -= 10  # lower fitness (failed)

                    dinosaurs.pop(j)

                    genomes.pop(j)

                    nets.pop(j)

        for i in rem\_list:

            enemies.pop(i)

            for j, dinosaur in enumerate(dinosaurs):

                genomes[j][1].fitness += 5  # raise fitness (+5 for every enemy)

        # controls

        for i, dinosaur in enumerate(dinosaurs):

            output = nets[i].activate((dinosaur.hitbox.y,

                                       calc\_dist((dinosaur.hitbox.x, dinosaur.hitbox.y), enemies[0].hitbox.midtop),

                                       enemies[0].hitbox.width,

                                       game\_speed))

            if output[0] > 0.5 and dinosaur.state is not DinoState.JUMP:

                dinosaur.jump()

                genomes[i][1].fitness -= 1  # every jump lowers the fitness (assuming it's false jump)

        # read user input (jump test)

        # user\_input = pygame.key.get\_pressed()

        # if user\_input[pygame.K\_SPACE]:

        #     for dino in dinosaurs:

        #         if not dino.state == DinoState.JUMP:

        #             dino.jump()

        # score & game speed

        score += 0.5 \* (game\_speed / 4)

        if score > score\_speedup:

            score\_speedup += 100 \* (game\_speed / 2)

            game\_speed += 1

            print(f"Game speed increased - {game\_speed}")

        score\_label = score\_font.render("Очки: " + str(math.floor(score)), True, (50, 50, 50))

        score\_label\_rect = score\_label.get\_rect()

        score\_label\_rect.center = (width - 100, 50)

        screen.blit(score\_label, score\_label\_rect)

        # display dinosaurs names

        for i, dinosaur in enumerate(dinosaurs):

            dname\_label = dname\_font.render(dinosaur.name, True, (170, 238, 187))

            dname\_label\_rect = dname\_label.get\_rect()

            dname\_label\_rect.center = (width - 100, 100 + (i \* 25))

            screen.blit(dname\_label, dname\_label\_rect)

        # display generation

        label = heading\_font.render("Поколение: " + str(generation), True, (0, 72, 186))

        label\_rect = label.get\_rect()

        label\_rect.center = (width / 2, 150)

        screen.blit(label, label\_rect)

        # display game speed

        score\_label = score\_font.render("Скорость: " + str(game\_speed / 8) + "x", True, (50, 50, 50))

        score\_label\_rect = score\_label.get\_rect()

        score\_label\_rect.center = (150, 50)

        screen.blit(score\_label, score\_label\_rect)

        # flip & tick

        pygame.display.flip()

        clock.tick(60)  # fixed 60 fps

if \_\_name\_\_ == "\_\_main\_\_":

    # setup config

    config\_path = "./config-feedforward.txt"

    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpeciesSet,

                                neat.DefaultStagnation, config\_path)

    # init NEAT

    p = neat.Population(config)

    # run NEAT

    p.run(run\_game, 1000)

**NEAT кітапханасының деректері:**

fitness\_criterion = max

fitness\_threshold = 10000

pop\_size = 20

reset\_on\_extinction = False

[DefaultGenome]

# node activation options

activation\_default = tanh

activation\_mutate\_rate = 0.01

activation\_options = tanh

# node aggregation options

aggregation\_default = sum

aggregation\_mutate\_rate = 0.01

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 = 0.5

# connection enable options

enabled\_default = True

enabled\_mutate\_rate = 0.01

feed\_forward = True

initial\_connection = full

# node add/remove rates

node\_add\_prob = 0.2

node\_delete\_prob = 0.2

# network parameters

num\_hidden = 0

num\_inputs = 4  **// Бұл ойын қабылдайтын параметрлер яғни кактус биіктігі ені арақашықтығы жылдамдық**

num\_outputs = 1 **// Ойын шешім қабылдап бір ғана аргументті қайтарады ол секіру (jump) функциясы**

# node response options

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

# 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 = 20

species\_elitism = 2

[DefaultReproduction]

elitism = 3

survival\_threshold = 0.2