**НИТУ «МИСиС»**

Институт ИТКН

**Кафедра инженерной кибернетики**

Направление подготовки: 01.03.04 прикладная математика

Квалификация (степень): бакалавр

**КУРСОВАЯ РАБОТА**

**учебная дисциплина**

**«Искусственные нейронные сети»**

**VII семестр 2020 – 2021 у.г.**

**Учащийся: Ипатов Д.Д.**

**Группа: БПМ-17-2**

**Оценка:**

**Дата защиты:**

**Москва, 2020 г.**

**СОДЕРЖАНИЕ**

[Теоретическая справка 3](#_Toc59376616)

[Использованные средства разработки 7](#_Toc59376617)

[Описание генетического алгоритма. 9](#_Toc59376618)

[Разработанное программное обеспечение 11](#_Toc59376619)

[Вывод 14](#_Toc59376620)

[Список использованных источников 15](#_Toc59376621)

[Приложения 16](#_Toc59376622)

# **Теоретическая справка**

«**Тетрис**» представляет собой головоломку, построенную на использовании геометрических фигур «**тетрамино**» — разновидности полимино, состоящих из четырёх квадратов

Случайные фигурки тетрамино падают сверху в прямоугольный стакан шириной 10 и высотой 20 клеток. В полёте игрок может поворачивать фигурку на 90° и двигать её по горизонтали. Также можно «сбрасывать» фигурку, то есть ускорять её падение, когда уже решено, куда фигурка должна упасть. Фигурка летит до тех пор, пока не наткнётся на другую фигурку либо на дно стакана. Если при этом заполнился горизонтальный ряд из 10 клеток, он пропадает и всё, что выше него, опускается на одну клетку. Дополнительно показывается фигурка, которая будет следовать после текущей — это подсказка, которая позволяет игроку планировать действия. Темп игры постепенно ускоряется. Игра заканчивается, когда новая фигурка не может поместиться в стакан. Игрок получает очки за каждый заполненный ряд, поэтому его задача — заполнять ряды, не заполняя сам стакан (по вертикали) как можно дольше, чтобы таким образом получить как можно больше очков [1].

**Обучение с подкреплением (Q-Learning)** — это метод машинного обучения, при котором происходит обучение модели, которая не имеет сведений о системе, но имеет возможность производить какие-либо действия в ней. Действия переводят систему в новое состояние и модель получает от системы некоторое вознаграждение [2].

**Однослойная структура нейронной сети**. Представляет собой структуру взаимодействия нейронов, в которой сигналы со входного слоя сразу направляются на выходной слой, который, собственно говоря, не только преобразует сигнал, но и сразу же выдаёт ответ. Как уже было сказано, 1-й входной слой только принимает и распределяет сигналы, а нужные вычисления происходят уже во втором слое. Входные нейроны являются объединёнными с основным слоем с помощью синапсов с разными весами, обеспечивающими качество связей.

**Многослойная нейронная сеть**. Здесь, помимо выходного и входного слоёв, имеются ещё несколько скрытых промежуточных слоёв. Число этих слоёв зависит от степени сложности нейронной сети. Она в большей степени напоминает структуру биологической нейронной сети. Такие виды были разработаны совсем недавно, до этого все процессы были реализованы с помощью однослойных нейронных сетей. Соответствующие решения обладают большими возможностями, если сравнивать с однослойными, ведь в процессе обработки данных каждый промежуточный слой — это промежуточный этап, на котором осуществляется обработка и распределение информации.

**Нейросети прямого распространения (однонаправленные)**. В этой структуре сигнал перемещается строго по направлению от входного слоя к выходному. Движение сигнала в обратном направлении не осуществляется и в принципе невозможно. Сегодня разработки этого плана распространены широко и на сегодняшний день успешно решают задачи распознавания образов, прогнозирования и кластеризации [3].

**Генетический алгоритм** — это в первую очередь эволюционный алгоритм, другими словами, основная фишка алгоритма — скрещивание (комбинирование). Как несложно догадаться идея алгоритма наглым образом взята у природы, благо она не подаст на это в суд. Так вот, путем перебора и самое главное отбора получается правильная «комбинация».  
Алгоритм делится на три этапа:

* Скрещивание
* Селекция (отбор)
* Формирования нового поколения

Если результат нас не устраивает, эти шаги повторяются до тех пор, пока результат нас не начнет удовлетворять или произойдет одно из нижеперечисленных условий:

* Количество поколений (циклов) достигнет заранее выбранного максимума
* Исчерпано время на мутацию [4].

**Постановка задачи**

Разработать прототип нейронной сети, позволяющей играть в Тетрис, и ПО для игры в Тетрис, протестировать на нем полученную нейронную сеть.

Этапы выполнения задачи:

* 1. Выбрать класс нейронной сети;
  2. Выбрать метод обучения нейронной сети;
  3. Разработать генетический алгоритм обучения на C++;
  4. Обучить нейронную сеть;
  5. Написать ПО для игры в Тетрис;
  6. Написать ИИ для игры в Тетрис;
  7. Протестировать ИИ на игре в Тетрис.

# **Использованные средства разработки**

Технологический стек:

* 1. Для реализации нейронной сети и дальнейшего обучения был выбран язык C++.
  2. Среда разработки – Microsoft Visual Studio 2019
  3. При разработке на C++ использовалась библиотека STL
  4. Для разработки ПО для игры в Тетрис был выбран язык Python 3.
  5. Среда разработки – IDLE
  6. При разработке на Python 3 были использованы следующие библиотеки: Numpy, PyGame, TensorFlow

**Описание нейронной сети**

Была выбрана многослойная сеть прямого распространения со следующими характеристиками:

* Количество входных нейронов: 5.
* Количество выходных нейронов: 1.
* Количество скрытых слоев: 1.
* Функция активации: ReLU.
* Количество нейронов на скрытом

На вход сети приходили следующие переменные:

1. Суммарная разница в высоте столбцов;
2. Количество «дырок» (пустые клетки на поле, в которые невозможно установить тетрамино);
3. Число линей (полностью заполненные ряды на поле);
4. Максимальная высота столбца;
5. Минимальная высота столбца.

В результате обработки нейронной сетью входных переменных выдавалась оценка «состояния», то есть текущая тетрамино помещалась во все возможные конечные положения (с учетом поворота тетрамино) на поле, а затем для каждого такого состояния считалась его оценка. В результате выбиралось состояние с наибольшей оценкой, в которое помещалось тетрамино в дальнейшем.

**Описание генетического алгоритма.**

Для подбора параметров ИНС был выбран генетический алгоритм. Этапы генетического алгоритма представлены на рисунке 1.

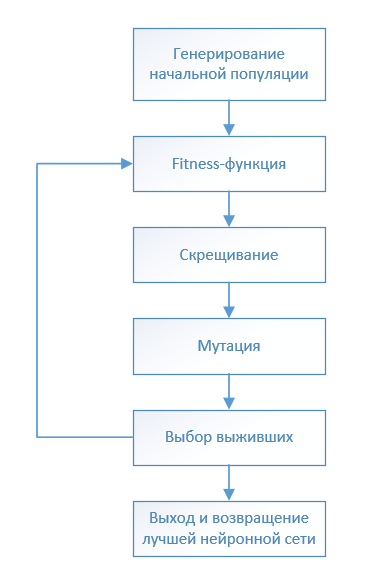


Рисунок 1 – Схема генетического алгоритма

Поэтому весь генетический алгоритм можно разбить на следующие этапы:

1. Генерирование начальной популяции: на данном этапе создалось 200 начальных нейронных сетей со случайными параметрами, взятыми из равномерного распределения на отрезке [-1; 1];
2. Fitness-функция: на данном этапе происходил подсчет эффективности каждой нейронной сети, основной параметр оценки – количество удаленных линий за игру;
3. Скрещивания: происходило слияние 100 случайных лучших нейронных сетей, в результате чего получалось 100 потомков. Скрещивание происходило следующим образом: равновероятно выбиралось 2 из 100 лучших нейронных сетей, затем происходило заполнение параметров новой нейронной сети, в которой каждый параметр выбирался равновероятно у одного из родителей;
4. Мутация: далее у каждой нейронной сети каждый параметр с вероятность 0.05 заменялся на новый параметр, взятый из нормального стандартного распределения;
5. Выбор выживших: на данном этапе 100 худших нейронный сетей заменялись новыми, полученными в результате скрещивания.

Данные шаги повторялись 100 раз, то есть обработалось 100 эпох. В результате работы была получена лучшая нейронная сеть, которая затем являлась базисом для дальнейшего ИИ для игры в Тетрис.

На рисунках 2 и 3 можно видеть примеры эпох с весами параметров нейронной сети на данной стадии.

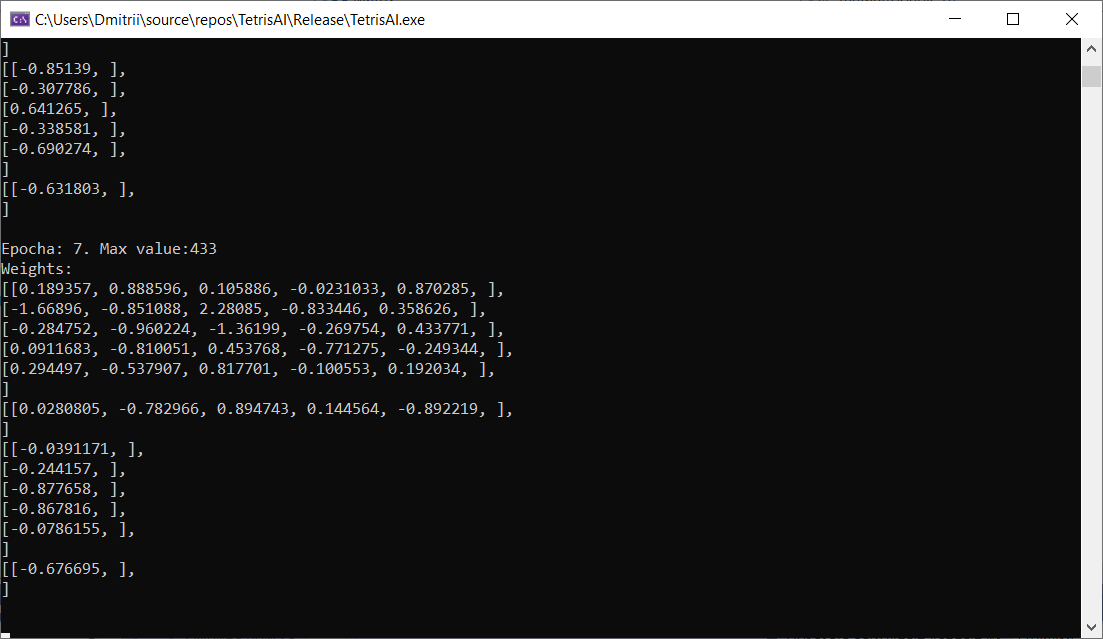


Рисунок 2 – Эпоха 7

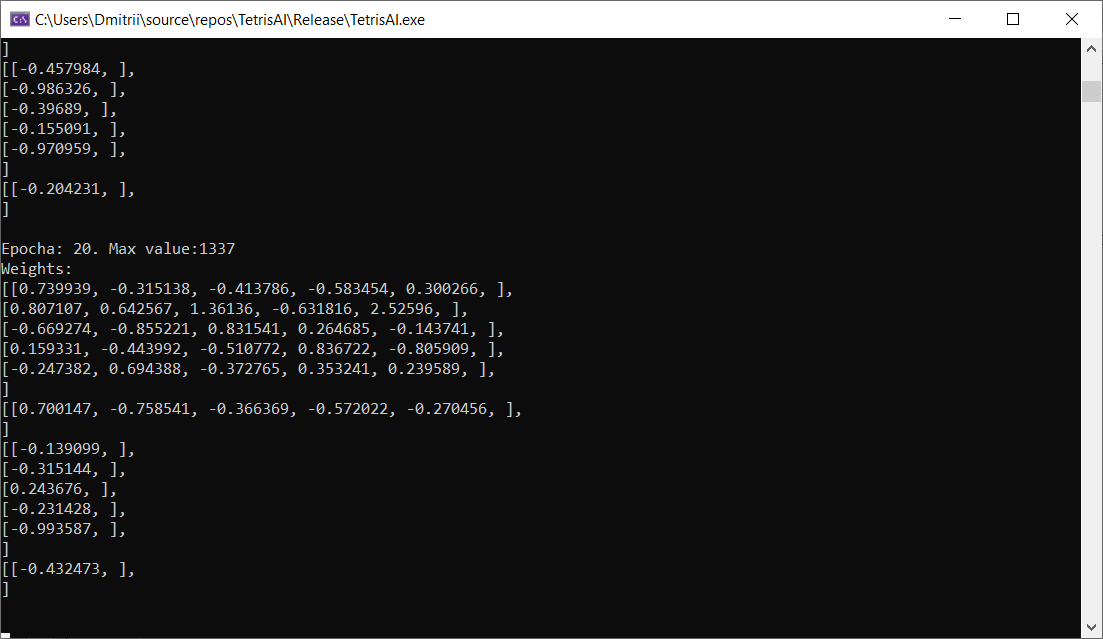


Рисунок – Эпоха 20

# **Разработанное программное обеспечение**

**Общие сведения**

Разработанное программное обеспечение предполагает использование его в некоммерческих, образовательных целях.

**Основные экранные формы, иллюстрирующие работу системы**

При запуске приложения открывается окно ожидания, после нажатие любой клавиши клавиатуры откроется основное окно приложения.

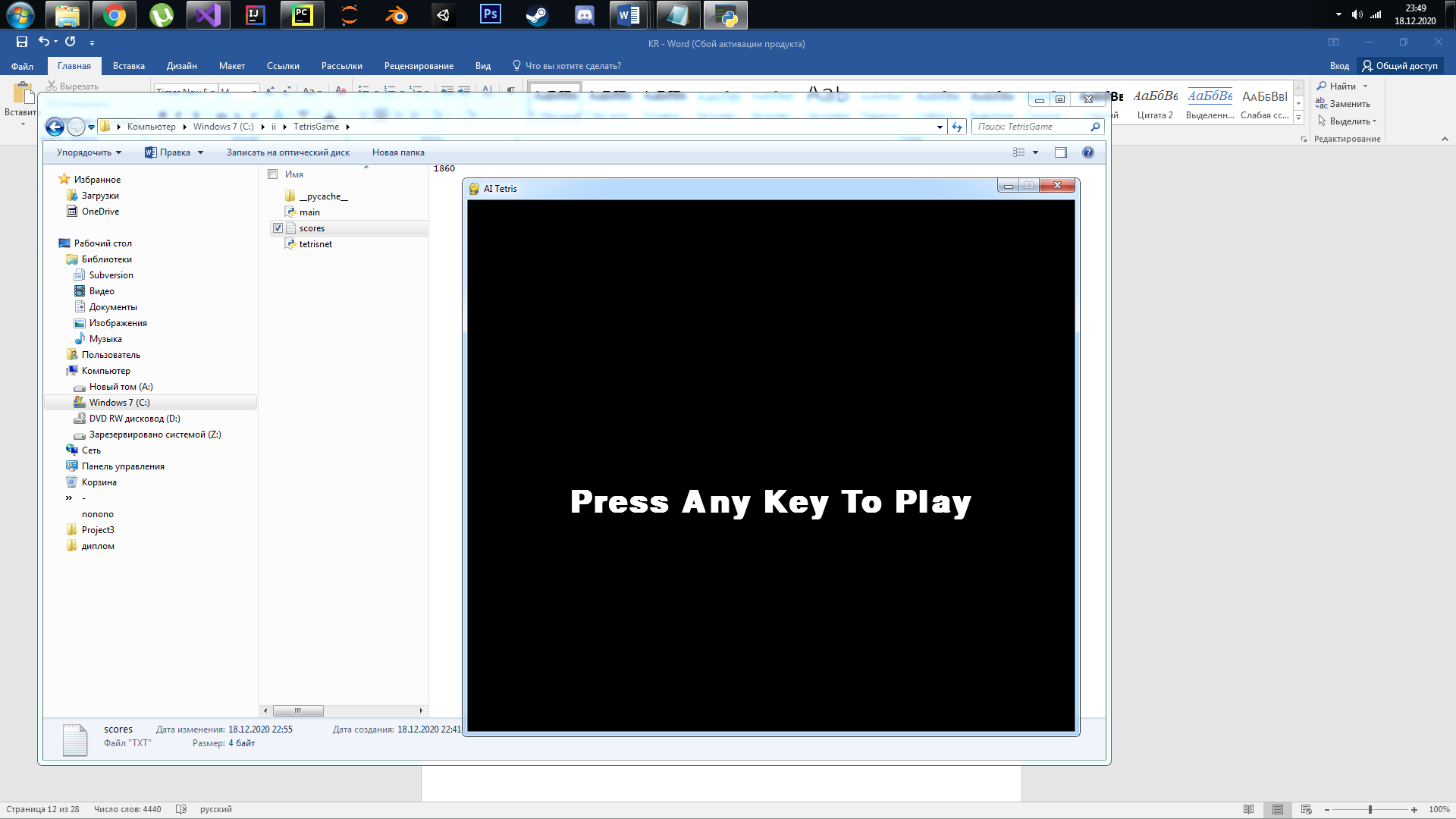


Рисунок 4 – Окно запуска приложения

На следующем рисунке изображен промежуточный результат работы приложения. В основном окне отображается следующая случайная фигура тетрамино, текущее количество очков, максимально количество очков.

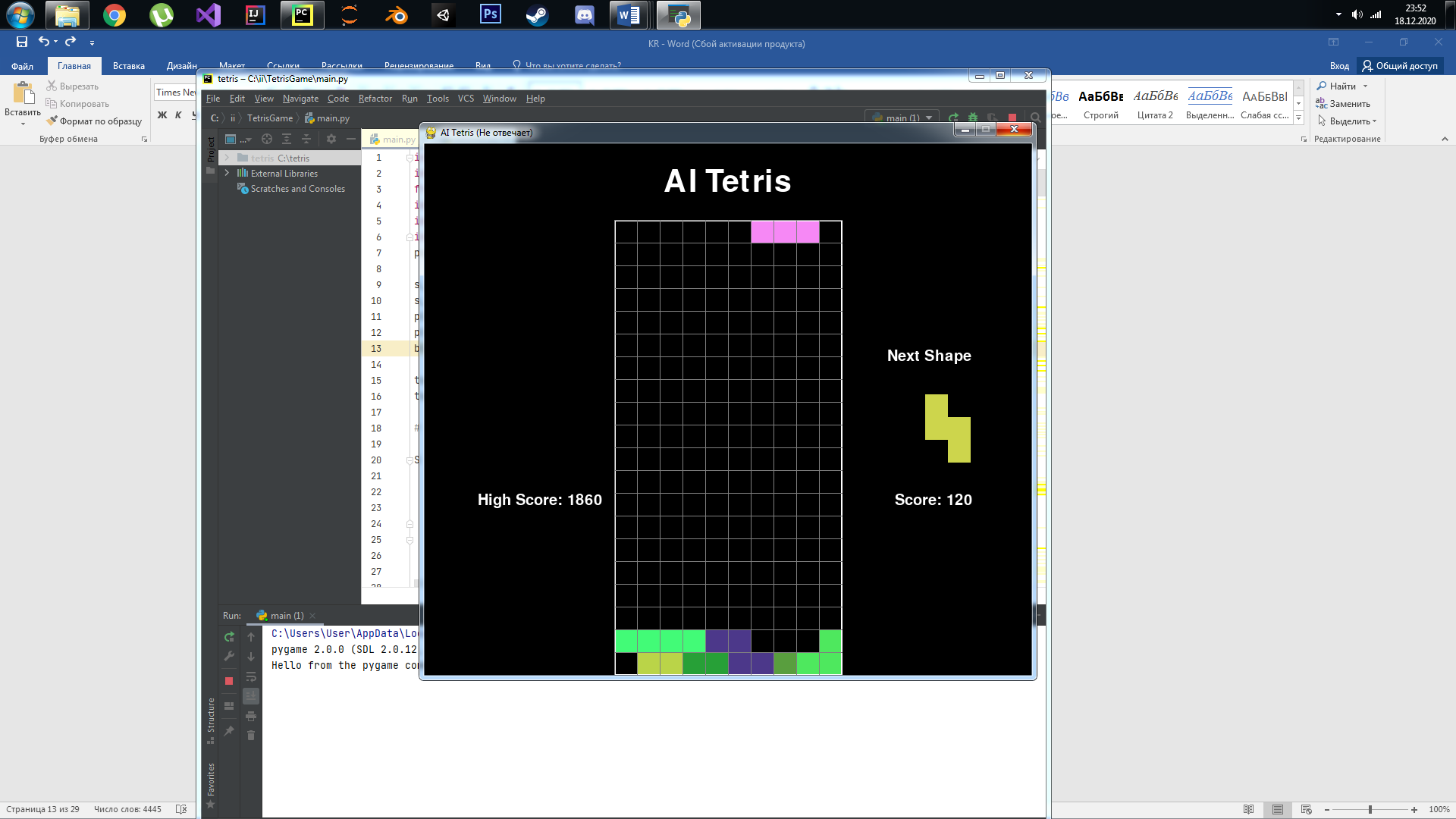


Рисунок 5 – Пример работы приложения

На рисунке 6 представлен финальный результат работы приложения. В результате работы программы было удалено 85 строк. Максимальный результат работы программы остался прежним – 186 строк.

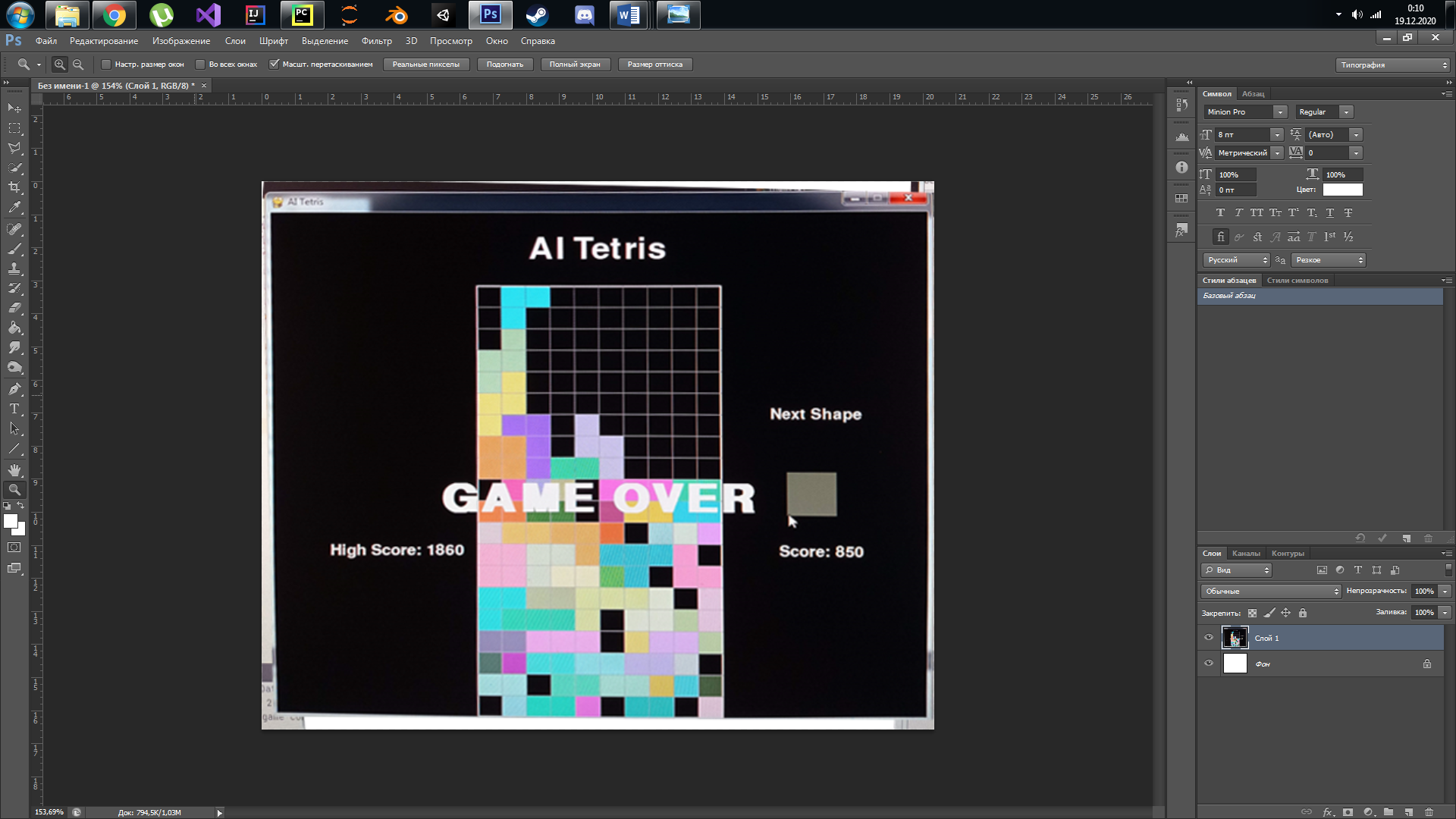


Рисунок 6 – Результат работы приложения

# **ВЫВОД**

В результате выполнения курсовой работы было разработано ИИ на базе нейронных сетей для игры в Тетрис. Удалось добиться результата у ИИ в 186 уничтоженных линий за игру.

В процессе работы был изучен генетический алгоритм, который помогают решать задачи обучения с подкреплением. А также удалось немного побольше познакомиться с нейронными сетями.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. <https://ru.wikipedia.org/wiki/Тетрис>
2. <https://habr.com/ru/post/308094/>
3. <https://otus.ru/nest/post/1263/>
4. <https://habr.com/ru/post/128704/>

# **ПРИЛОЖЕНИЯ**

Приложение 1. Код для игры в Тетрис. Файл main.py

import pygame

import random

from random import choice, randrange

import tetrisnet

import numpy as np

import torch

pygame.font.init()

s\_width = 800

s\_height = 700

play\_width = 300

play\_height = 600

block\_size = 30

top\_left\_x = (s\_width - play\_width) // 2

top\_left\_y = s\_height - play\_height

# shape formats

S = [['.....',

'.....',

'..00.',

'.00..',

'.....'],

['.....',

'..0..',

'..00.',

'...0.',

'.....']]

Z = [['.....',

'.....',

'.00..',

'..00.',

'.....'],

['.....',

'..0..',

'.00..',

'.0...',

'.....']]

I = [['..0..',

'..0..',

'..0..',

'..0..',

'.....'],

['.....',

'0000.',

'.....',

'.....',

'.....']]

O = [['.....',

'.....',

'.00..',

'.00..',

'.....']]

J = [['.....',

'.0...',

'.000.',

'.....',

'.....'],

['.....',

'..00.',

'..0..',

'..0..',

'.....'],

['.....',

'.....',

'.000.',

'...0.',

'.....'],

['.....',

'..0..',

'..0..',

'.00..',

'.....']]

L = [['.....',

'...0.',

'.000.',

'.....',

'.....'],

['.....',

'..0..',

'..0..',

'..00.',

'.....'],

['.....',

'.....',

'.000.',

'.0...',

'.....'],

['.....',

'.00..',

'..0..',

'..0..',

'.....']]

T = [['.....',

'..0..',

'.000.',

'.....',

'.....'],

['.....',

'..0..',

'..00.',

'..0..',

'.....'],

['.....',

'.....',

'.000.',

'..0..',

'.....'],

['.....',

'..0..',

'.00..',

'..0..',

'.....']]

shapes = [S, Z, I, O, J, L, T]

#shape\_colors = [(0, 255, 0), (255, 0, 0), (0, 255, 255), (255, 255, 0), (255, 165, 0), (0, 0, 255), (128, 0, 128)]

shape\_colors = lambda : (randrange(30, 256), randrange(30, 256), randrange(30, 256))

class Piece(object):

def \_\_init\_\_(self, x, y, shape):

self.x = x

self.y = y

self.shape = shape

self.color = shape\_colors()

self.rotation = random.randint(0,len(shape) - 1)

def create\_grid(locked\_pos={}):

grid = [[(0,0,0) for \_ in range(10)] for \_ in range(20)]

for i in range(len(grid)):

for j in range(len(grid[i])):

if (j, i) in locked\_pos:

c = locked\_pos[(j,i)]

grid[i][j] = c

return grid

def convert\_shape\_format(shape):

positions = []

format = shape.shape[shape.rotation % len(shape.shape)]

for i, line in enumerate(format):

row = list(line)

for j, column in enumerate(row):

if column == '0':

positions.append((shape.x + j, shape.y + i))

for i, pos in enumerate(positions):

positions[i] = (pos[0] - 2, pos[1] - 4)

return positions

def valid\_space(shape, grid):

accepted\_pos = [[(j, i) for j in range(10) if grid[i][j] == (0,0,0)] for i in range(20)]

accepted\_pos = [j for sub in accepted\_pos for j in sub]

formatted = convert\_shape\_format(shape)

for pos in formatted:

if pos not in accepted\_pos:

if pos[1] > -1:

return False

return True

def check\_lost(positions):

for pos in positions:

x, y = pos

if y < 1:

return True

return False

def get\_shape():

return Piece(5, 0, random.choice(shapes))

def draw\_text\_middle(surface, text, size, color):

font = pygame.font.SysFont("comicsans", size, bold=True)

label = font.render(text, 1, color)

surface.blit(label, (top\_left\_x + play\_width /2 - (label.get\_width()/2), top\_left\_y + play\_height/2 - label.get\_height()/2))

def draw\_grid(surface, grid):

sx = top\_left\_x

sy = top\_left\_y

for i in range(len(grid)):

pygame.draw.line(surface, (128, 128, 128), (sx, sy + i\*block\_size), (sx+play\_width, sy+ i\*block\_size))

for j in range(len(grid[i])):

pygame.draw.line(surface, (128, 128, 128), (sx + j\*block\_size, sy),(sx + j\*block\_size, sy + play\_height))

def clear\_rows(grid, locked):

inc = 0

for i in range(len(grid)-1, -1, -1):

row = grid[i]

if (0,0,0) not in row:

inc += 1

ind = i

for j in range(len(row)):

try:

del locked[(j,i)]

except:

continue

if inc > 0:

for key in sorted(list(locked), key=lambda x: x[1])[::-1]:

x, y = key

if y < ind:

newKey = (x, y + inc)

locked[newKey] = locked.pop(key)

return inc

def draw\_next\_shape(shape, surface):

font = pygame.font.SysFont('comicsans', 30)

label = font.render('Next Shape', 1, (255,255,255))

sx = top\_left\_x + play\_width + 50

sy = top\_left\_y + play\_height/2 - 100

format = shape.shape[shape.rotation % len(shape.shape)]

for i, line in enumerate(format):

row = list(line)

for j, column in enumerate(row):

if column == '0':

pygame.draw.rect(surface, shape.color, (sx + j\*block\_size, sy + i\*block\_size, block\_size, block\_size), 0)

surface.blit(label, (sx + 10, sy - 30))

def update\_score(nscore):

score = max\_score()

with open('scores.txt', 'w') as f:

if int(score) > nscore:

f.write(str(score))

else:

f.write(str(nscore))

def max\_score():

with open('scores.txt', 'r') as f:

lines = f.readlines()

score = lines[0].strip()

return score

def draw\_window(surface, grid, score=0, last\_score = 0):

surface.fill((0, 0, 0))

pygame.font.init()

font = pygame.font.SysFont('comicsans', 60)

label = font.render('AI Tetris', 1, (255, 255, 255))

surface.blit(label, (top\_left\_x + play\_width / 2 - (label.get\_width() / 2), 30))

# current score

font = pygame.font.SysFont('comicsans', 30)

label = font.render('Score: ' + str(score), 1, (255,255,255))

sx = top\_left\_x + play\_width + 50

sy = top\_left\_y + play\_height/2 - 100

surface.blit(label, (sx + 20, sy + 160))

# last score

label = font.render('High Score: ' + last\_score, 1, (255,255,255))

sx = top\_left\_x - 200

sy = top\_left\_y + 200

surface.blit(label, (sx + 20, sy + 160))

for i in range(len(grid)):

for j in range(len(grid[i])):

pygame.draw.rect(surface, grid[i][j], (top\_left\_x + j\*block\_size, top\_left\_y + i\*block\_size, block\_size, block\_size), 0)

pygame.draw.rect(surface, (255, 255, 255), (top\_left\_x, top\_left\_y, play\_width, play\_height), 2)

draw\_grid(surface, grid)

def get\_fig(fig):

min\_x = 100

min\_y = 100

max\_x = -1

max\_y = -1

for i in range(len(fig)):

for j in range(len(fig[0])):

if fig[i][j] == 1:

min\_x = min(j,min\_x)

min\_y = min(i,min\_y)

max\_x = max(j,max\_x)

max\_y = max(i,max\_y)

return fig[min\_y:max\_y+1,min\_x:max\_x+1].copy()

def get\_field\_with\_fig(field, fig, i):

max\_y = -1

for j in range(0,len(field)-len(fig)+1):

flag = True

for y in range(len(fig)):

for x in range(len(fig[0])):

if field[j+y][x+i] == 1 and fig[y][x] == 1:

flag = False

if flag:

max\_y = j

else:

break

if max\_y == -1:

return field.copy()

else:

temp\_field = field.copy()

for y in range(len(fig)):

for x in range(len(fig[0])):

if fig[y][x] == 1:

temp\_field[y+max\_y][x+i] = 1

return temp\_field

def get\_height(field, i):

for j in range(len(field)):

if field[j][i] == 1:

return len(field) - j

return 0

def get\_differences(heights):

return sum([abs(heights[i] - heights[i+1]) for i in range(len(heights)-1)])

def get\_amount\_lines(field):

nums = 0

for j in range(len(field)):

flag = True

for i in range(len(field[0])):

if field[j][i] == 0:

flag = False

break

if flag:

nums += 1

return nums

def get\_holes(field, heights):

holes = 0

for i in range(len(heights)):

for j in range(len(field) - heights[i], len(field)):

if(field[j][i] == 0):

holes += 1

return holes

def get\_step(grid, current\_piece, TNN):

scores = -1e10

x = 0

rota = 0

for rot in range(len(current\_piece.shape)):

field = np.zeros((20,10))

for i in range(20):

for j in range(10):

if grid[i][j] == (0,0,0):

field[i][j] = 0.

else:

field[i][j] = 1.

fig = np.zeros((5,5))

for i in range(5):

for j in range(5):

if current\_piece.shape[rot][i][j] == '0':

fig[i][j] = 1.

else:

fig[i][j] = 0.

fig = get\_fig(fig)

for i in range(len(field[0])-len(fig[0])+1):

temp\_field = get\_field\_with\_fig(field,fig,i)

heights = [get\_height(temp\_field, j) for j in range(len(temp\_field[0]))]

differences = get\_differences(heights)

amount\_lines = get\_amount\_lines(temp\_field)

holes = get\_holes(temp\_field, heights)

max\_height = max(heights)

min\_height = min(heights)

score = TNN.forward(np.array([float(differences), float(holes), float(amount\_lines), float(min\_height), float(max\_height)]))

if score > scores:

x = i

rota = rot

scores = score

return x, rota

def get\_current\_pos(current\_piece):

if current\_piece.shape == S:

if current\_piece.rotation == 0:

return 4

else:

return 5

elif current\_piece.shape == I:

if current\_piece.rotation == 0:

return 5

else:

return 3

elif current\_piece.shape == J:

if current\_piece.rotation == 1:

return 5

else:

return 4

elif current\_piece.shape == L:

if current\_piece.rotation == 1:

return 5

else:

return 4

elif current\_piece.shape == T:

if current\_piece.rotation == 1:

return 5

else:

return 4

else:

return 4

def main(win):

last\_score = max\_score()

locked\_positions = {}

grid = create\_grid(locked\_positions)

TNN = tetrisnet.TetrisNet()

TNN.m1 = np.matrix([[-0.874313, 0.302529, -1.1421, -0.257178, -0.869937, ],

[1.13769, 1.6891, -0.0688631, -1.35793, -1.66535, ],

[0.0370188, -0.515627, -1.25191, 0.627983, -0.532746, ],

[-0.0497609, -0.371445, -0.89942, 2.01398, 2.18431, ],

[-0.350091, 0.0531693, -1.6935, -1.37114, 0.13328, ],

])

TNN.bias1 = np.matrix([[0.299143, 0.745956, -0.769967, -0.155007, -0.149685, ],

])

TNN.m2 = np.matrix([[-0.304138, ],

[-1.89745, ],

[0.43527, ],

[0.0803377, ],

[-0.0400605, ],

])

TNN.bias2 = np.matrix([[0.476275, ],

])

change\_piece = False

run = True

current\_piece = get\_shape()

next\_piece = get\_shape()

clock = pygame.time.Clock()

fall\_time = 0

change\_time = 0

fall\_speed = 0.27

level\_time = 0

score = 0

grid = create\_grid(locked\_positions)

step, rota = get\_step(grid, current\_piece, TNN)

current\_pos\_x = get\_current\_pos(current\_piece)

flag = 0

while run:

grid = create\_grid(locked\_positions)

fall\_time += clock.get\_rawtime()

level\_time += clock.get\_rawtime()

change\_time += clock.get\_rawtime()

clock.tick()

if level\_time/1000 > 5:

level\_time = 0

if level\_time > 0.12:

level\_time -= 0.005

if fall\_time/1000 > fall\_speed:

fall\_time = 0

current\_piece.y += 1

flag += 1

if not(valid\_space(current\_piece, grid)) and current\_piece.y > 0:

current\_piece.y -= 1

change\_piece = True

if change\_time/1000 > fall\_speed/30:

change\_time = 0

if current\_piece.rotation != rota:

up(current\_piece, grid)

current\_pos\_x = get\_current\_pos(current\_piece)

elif current\_pos\_x > step:

current\_pos\_x -=1

left(current\_piece, grid)

elif current\_pos\_x < step:

current\_pos\_x += 1

right(current\_piece, grid)

else:

down(current\_piece, grid)

shape\_pos = convert\_shape\_format(current\_piece)

for i in range(len(shape\_pos)):

x, y = shape\_pos[i]

if y > -1:

grid[y][x] = current\_piece.color

if change\_piece:

for pos in shape\_pos:

p = (pos[0], pos[1])

locked\_positions[p] = current\_piece.color

current\_piece = next\_piece

next\_piece = get\_shape()

flag = 0

change\_piece = False

score += clear\_rows(grid, locked\_positions) \* 10

grid = create\_grid(locked\_positions)

step, rota = get\_step(grid, current\_piece, TNN)

current\_pos\_x = get\_current\_pos(current\_piece)

draw\_window(win, grid, score, last\_score)

draw\_next\_shape(next\_piece, win)

pygame.display.update()

if check\_lost(locked\_positions):

draw\_text\_middle(win, "GAME OVER", 80, (255,255,255))

pygame.display.update()

pygame.time.delay(1500)

run = False

update\_score(score)

def right(current\_piece, grid):

current\_piece.x += 1

if not (valid\_space(current\_piece, grid)):

current\_piece.x -= 1

def left(current\_piece, grid):

current\_piece.x -= 1

if not (valid\_space(current\_piece, grid)):

current\_piece.x += 1

def down(current\_piece, grid):

current\_piece.y += 1

if not (valid\_space(current\_piece, grid)):

current\_piece.y -= 1

def up(current\_piece, grid):

current\_piece.rotation = (current\_piece.rotation+1)%(len(current\_piece.shape))

if not (valid\_space(current\_piece, grid)):

current\_piece.rotation = (len(current\_piece.shape)+current\_piece.rotation-1)%(len(current\_piece.shape))

def main\_menu(win): # \*

run = True

while run:

win.fill((0,0,0))

draw\_text\_middle(win, 'Press Any Key To Play', 60, (255,255,255))

pygame.display.update()

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if event.type == pygame.KEYDOWN:

main(win)

pygame.display.quit()

win = pygame.display.set\_mode((s\_width, s\_height))

pygame.display.set\_caption('AI Tetris')

main\_menu(win)

Приложение 2. Нейронная сеть на Python 3 . Файл tetrisnet.py

import torch

import numpy as np

from torch import nn

class TetrisNet ():

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

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

self.m1 = np.array([])

self.bias1 = np.array([])

self.m2 = np.array([])

self.bias2 = np.array([])

def forward(self, x):

x = x.dot(self.m1)

x += self.bias1

for i in range(len(x[0])):

x[0, i] = max(x[0,i], 0.)

x = x.dot(self.m2)

x += self.bias2

return x[0][0]

Приложение 3. Матрица с основными функциями. Файл Matrix.h

#pragma once

#include <vector>

#include <random>

using namespace std;

struct Matrix {

vector<vector<double>> matr;

Matrix() = default;

Matrix(int rows, int columns)

: matr(rows, vector<double>(columns))

{

}

const Matrix dot(const Matrix& r) {

Matrix result(matr.size(), r.matr[0].size());

for (int i = 0; i < matr.size(); ++i) {

for (int j = 0; j < r.matr[0].size(); ++j) {

double sum = 0;

for (int k = 0; k < matr[0].size(); ++k) {

sum += matr[i][k] \* r.matr[k][j];

}

result.matr[i][j] = sum;

}

}

return result;

}

void randomize() {

random\_device randomDevice;

mt19937 randomGen(randomDevice());

uniform\_real\_distribution<double> distribution(-1, 1);

for (int i = 0; i < matr.size(); ++i) {

for (int j = 0; j < matr[0].size(); ++j) {

matr[i][j] = distribution(randomGen);

}

}

}

void mutant(double p) {

random\_device randomDevice;

mt19937 randomGen(randomDevice());

normal\_distribution<double> norm(0, 1);

bernoulli\_distribution b(p);

for (int i = 0; i < rows(); ++i) {

for (int j = 0; j < columns(); ++j) {

if (b(randomGen)) {

matr[i][j] = norm(randomGen);

}

}

}

}

const int rows() {

return matr.size();

}

const int columns() {

return matr[0].size();

}

Matrix& operator= (const Matrix& r)

{

matr = r.matr;

return \*this;

}

};

Приложение 4. Обучение нейронной сети. Файл Source.cpp

#include <iostream>

#include "Matrix.h"

#include "TetrisNet.h"

#include "Tetris.h"

#include <random>

#include <vector>

#include <algorithm>

using namespace std;

TetrisNet create\_new\_individual(int input, int hidden, int output) {

TetrisNet net(input, hidden, output);

net.randomize();

return net;

}

vector<TetrisNet> create\_individuals(int amount, int input, int hidden, int output) {

vector<TetrisNet> population;

for (int i = 0; i < amount; ++i) {

population.push\_back(create\_new\_individual(input, hidden, output));

}

return population;

}

void fitnesses(vector<TetrisNet>& individuals, Tetris& tetris) {

for (int i = 0; i < individuals.size(); ++i) {

if (!individuals[i].flag) {

tetris.fitness(individuals[i]);

}

}

}

int main() {

Tetris tetris;

int input = 5;

int hidden = 5;

int output = 1;

TetrisNet net(input, hidden, output);

net.randomize();

tetris.fitness(net);

int amount = 200;

double percentage\_survivors = 0.5;

double mutagen\_perc = 0.05;

double epochs = 100;

vector<TetrisNet> population = create\_individuals(amount, input, hidden, output);

for (int i = 0; i < epochs; ++i) {

fitnesses(population, tetris);

sort(population.rbegin(), population.rend());

cout << "Epocha: " << i << ". Max value:" << population[0].score << endl;

cout << "Weights:" << endl;

cout << "[";

for (int i = 0; i < population[0].m1.rows(); ++i) {

cout << "[";

for (int j = 0; j < population[0].m1.columns(); ++j) {

cout << population[0].m1.matr[i][j] << ", ";

}

cout << "]," << endl;

}

cout << "]" << endl;

cout << "[";

for (int i = 0; i < population[0].bias1.rows(); ++i) {

cout << "[";

for (int j = 0; j < population[0].bias1.columns(); ++j) {

cout << population[0].bias1.matr[i][j] << ", ";

}

cout << "]," << endl;

}

cout << "]" << endl;

cout << "[";

for (int i = 0; i < population[0].m2.rows(); ++i) {

cout << "[";

for (int j = 0; j < population[0].m2.columns(); ++j) {

cout << population[0].m2.matr[i][j] << ", ";

}

cout << "]," << endl;

}

cout << "]" << endl;

cout << "[";

for (int i = 0; i < population[0].bias2.rows(); ++i) {

cout << "[";

for (int j = 0; j < population[0].bias2.columns(); ++j) {

cout << population[0].bias2.matr[i][j] << ", ";

}

cout << "]," << endl;

}

cout << "]" << endl;

cout << endl;

int new\_surv = percentage\_survivors \* amount;

for (int i = 0; i < new\_surv; ++i) {

population.pop\_back();

}

int surv = amount - new\_surv;

while (population.size() < amount) {

random\_device randomDevice;

mt19937 randomGen(randomDevice());

uniform\_int\_distribution<> uni(0, surv-1);

int p1 = uni(randomGen);

int p2 = uni(randomGen);

TetrisNet net(input, hidden, output);

bernoulli\_distribution bern(0.5);

for (int i = 0; i < population[0].m1.rows(); ++i) {

for (int j = 0; j < population[0].m1.columns(); ++j) {

if (bern(randomGen)) {

net.m1.matr[i][j] = population[p1].m1.matr[i][j];

}

else {

net.m1.matr[i][j] = population[p2].m1.matr[i][j];

}

}

}

for (int i = 0; i < population[0].bias1.rows(); ++i) {

for (int j = 0; j < population[0].bias1.columns(); ++j) {

if (bern(randomGen)) {

net.bias1.matr[i][j] = population[p1].bias1.matr[i][j];

}

else {

net.bias1.matr[i][j] = population[p2].bias1.matr[i][j];

}

}

}

for (int i = 0; i < population[0].m2.rows(); ++i) {

for (int j = 0; j < population[0].m2.columns(); ++j) {

if (bern(randomGen)) {

net.m2.matr[i][j] = population[p1].m2.matr[i][j];

}

else {

net.m2.matr[i][j] = population[p2].m2.matr[i][j];

}

}

}

for (int i = 0; i < population[0].bias2.rows(); ++i) {

for (int j = 0; j < population[0].bias2.columns(); ++j) {

if (bern(randomGen)) {

net.bias2.matr[i][j] = population[p1].bias2.matr[i][j];

}

else {

net.bias2.matr[i][j] = population[p2].bias2.matr[i][j];

}

}

}

net.mutant(mutagen\_perc);

population.push\_back(net);

}

}

}

Приложение 5. Версия Тетриса без GUI на C++. Файл Tetris.h

#pragma once

#include <vector>

#include <iostream>

#include <random>

#include <algorithm>

#include "TetrisNet.h"

#include "Matrix.h"

using namespace std;

struct Tetris {

vector<vector<int>> field;

int score = 0;

const vector<vector<vector<int>>> I = {

{

{1,1,1,1}

},

{

{1},

{1},

{1},

{1}

}

};

const vector<vector<vector<int>>> T = {

{

{0,1,0},

{1,1,1}

},

{

{1,1,1},

{0,1,0}

},

{

{1,0},

{1,1},

{1,0}

},

{

{0,1},

{1,1},

{0,1}

}

};

const vector<vector<vector<int>>> S = {

{

{0,1,1},

{1,1,0}

},

{

{1,0},

{1,1},

{0,1}

}

};

const vector<vector<vector<int>>> Z = {

{

{1,1,0},

{0,1,1}

},

{

{0,1},

{1,1},

{1,0}

}

};

const vector<vector<vector<int>>> O = {

{

{1,1},

{1,1}

}

};

const vector<vector<vector<int>>> J = {

{

{1,0,0},

{1,1,1}

},

{

{1,1},

{1,0},

{1,0}

},

{

{1,1,1},

{0,0,1}

},

{

{0,1},

{0,1},

{1,1}

}

};

const vector<vector<vector<int>>> L = {

{

{0,0,1},

{1,1,1}

},

{

{1,0},

{1,0},

{1,1}

},

{

{1,1,1},

{1,0,0}

},

{

{1,1},

{0,1},

{0,1}

}

};

const vector< vector<vector<vector<int>>>> shape = { S, Z, I, O, J, L, T };

vector<vector<vector<int>>> current\_shape;

Tetris()

: field(20, vector<int>(10))

{}

void delete\_lines() {

for (int i = 0; i < 20; ++i) {

bool flag = true;

for (int j = 0; j < 10; ++j) {

if (field[i][j] == 0) {

flag = false;

break;

}

}

if (flag) {

score++;

for (int k = i - 1; k >= 0; k--) {

field[k + 1] = field[k];

}

field[0].assign(10, 0);

}

}

}

void new\_shape() {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(0, 6);

current\_shape = shape[distrib(gen)];

}

bool put\_at(vector<vector<int>>& field, int x, int y, int rot) {

if (y < 0) {

return false;

}

for (int i = y; i < y + current\_shape[rot].size(); ++i) {

for (int j = x; j < x + current\_shape[rot][0].size(); ++j) {

if (current\_shape[rot][i - y][j - x] == 1) {

field[i][j] = current\_shape[rot][i - y][j - x];

}

}

}

return true;

}

bool get\_field\_with\_fig(vector<vector<int>>& field, int rot, int i) {

int max\_y = -1;

for (int j = 0; j < 20 - current\_shape[rot].size() + 1; ++j) {

bool flag = true;

for (int y = 0; y < current\_shape[rot].size(); ++y) {

for (int x = 0; x < current\_shape[rot][0].size(); ++x) {

if (field[j + y][i + x] == 1 && current\_shape[rot][y][x] == 1) {

flag = false;

}

}

}

if (flag) {

max\_y = j;

}

else {

break;

}

}

if (max\_y == -1) {

return false;

}

else {

return put\_at(field, i, max\_y, rot);

}

}

int get\_height(vector<vector<int>>& field, int i) {

for (int j = 0; j < 20; ++j) {

if (field[j][i] == 1) {

return 20 - j;

}

}

return 0;

}

vector<int> get\_heights(vector<vector<int>>& field) {

vector<int> heights(10);

for (int i = 0; i < 10; ++i) {

heights[i] = get\_height(field, i);

}

return heights;

}

int get\_differences(vector<int>& heights) {

int sum = 0;

for (int i = 0; i < heights.size() - 1; ++i) {

sum += abs(heights[i] - heights[i + 1]);

}

return sum;

}

int get\_amount\_lines(vector<vector<int>>& field) {

int sum = 0;

for (int i = 0; i < 20; ++i) {

bool flag = true;

for (int j = 0; j < 10; ++j) {

if (field[i][j] == 0) {

flag = false;

break;

}

}

if (flag) sum++;

}

return sum;

}

int get\_holes(vector<vector<int>>& field, vector<int>&heights) {

int holes = 0;

for (int i = 0; i < 10; ++i) {

for (int j = 20 - heights[i]; j < 20; ++j) {

if (field[j][i] == 0) {

holes++;

}

}

}

return holes;

}

int get\_max\_height(vector<int>& heights) {

int m = -1;

for (int i = 0; i < 10; ++i) {

m = max(m, heights[i]);

}

return m;

}

int get\_min\_height(vector<int>& heights) {

int m = 100;

for (int i = 0; i < 10; ++i) {

m = min(m, heights[i]);

}

return m;

}

pair<int, int> get\_step(TetrisNet& net) {

double result = -1e9;

int x;

int rota;

for (int rot = 0; rot < current\_shape.size(); ++rot) {

for (int i = 0; i < 10 - current\_shape[rot][0].size() + 1; ++i) {

vector<vector<int>> temp\_field = field;

get\_field\_with\_fig(temp\_field, rot, i);

Matrix tem(1, 5);

vector<int> heights = get\_heights(temp\_field);

tem.matr[0][0] = get\_differences(heights);

tem.matr[0][1] = get\_holes(temp\_field, heights);

tem.matr[0][2] = get\_amount\_lines(temp\_field);

tem.matr[0][3] = get\_min\_height(heights);

tem.matr[0][4] = get\_max\_height(heights);

double res = net.forward(tem).matr[0][0];

if (res > result) {

result = res;

x = i;

rota = rot;

}

}

}

return {rota, x};

}

double fitness(TetrisNet& net, bool flag = false) {

field.assign(20, vector<int>(10, 0));

score = 0;

while (true) {

new\_shape();

pair<int, int> step = get\_step(net);

if (!get\_field\_with\_fig(field, step.first, step.second)) {

break;

}

delete\_lines();

if (flag) {

for (int i = 0; i < 20; ++i) {

for (int j = 0; j < 10; ++j) {

cout << field[i][j];

}

cout << endl;

}

cout << endl;

}

}

net.score = score;

return score;

}

};

Приложение 5. Нейронная сеть на C++. Файл TetrisNet.h

#pragma once

#include "Matrix.h"

#include <algorithm>

using namespace std;

struct TetrisNet {

Matrix m1, m2, bias1, bias2;

bool flag = false;

int score = 0;

TetrisNet(int input, int hidden, int output)

: m1(input, hidden),

m2(hidden, output),

bias1(1, hidden),

bias2(1, output)

{

}

TetrisNet& operator=(const TetrisNet& right) {

if (this == &right) {

return \*this;

}

m1 = right.m1;

m2 = right.m2;

bias1 = right.bias1;

bias2 = right.bias2;

flag = right.flag;

score = right.score;

return \*this;

}

void randomize() {

m1.randomize();

m2.randomize();

bias1.randomize();

bias2.randomize();

}

void mutant(double p) {

m1.mutant(p);

m2.mutant(p);

}

const Matrix forward(Matrix& in) {

Matrix temp = in.dot(m1);

for (int i = 0; i < temp.columns(); ++i) {

temp.matr[0][i] += bias1.matr[0][i];

}

temp = relu(temp);

temp = temp.dot(m2);

for (int i = 0; i < temp.columns(); ++i) {

temp.matr[0][i] += bias2.matr[0][i];

}

return temp;

}

const Matrix relu(Matrix in) {

Matrix out(in.rows(), in.columns());

for (int i = 0; i < in.rows(); ++i) {

for (int j = 0; j < in.columns(); ++j) {

out.matr[i][j] = max(0., in.matr[i][j]);

}

}

return out;

}

bool operator< (const TetrisNet& a) const {

return score < a.score;

}

};