

Министерство образования и науки РФ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Омский государственный технический университет»

Кафедра «Информатика и вычислительная техника»

Дисциплина
«Проектирование и тестирование
программного обеспечения»
Расчётно-графическая работа
на тему
«Генетический алгоритм, простая игра»

Выполнил

Студент гр. ПИН-221

Шадчиев Г.А.

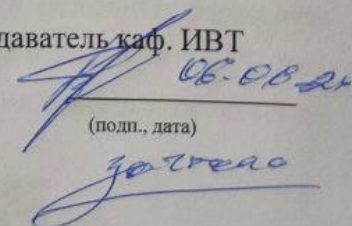


(подп., дата)

Проверил

Старший преподаватель каф. ИВТ

Блохин А.В.



(подп., дата)

Омск, 2024

ОСНОВНАЯ ЧАСТЬ

Часть 1. Описание задачи

В данной расчётно-графической работе приводится пример генетического алгоритма для создания искусственного интеллекта для игры. Два игрока по очереди делают ходы на доске 4x4 клетки. Можно перейти на любую из соседних клеток, пытающийся выйти за ее пределы игрок пропускает ход. Цель игры – в свой ход перейти на ту позицию, на которой сейчас находится соперник. Ограничение – нельзя 2 раза подряд ходить в одном направлении, иначе засчитано поражение.

Часть 2. Принцип работы и алгоритм:

Программа использует генетический алгоритм для выявления лучшего игрока, путем скрещивания и мутации хромосом, состоящих из значений типа double.

Используются такие библиотеки как: random для определения вероятности скрещивания и мутации, numpy для удобного и быстрого пользования массивами, math для вычисления по формуле расстояния между координатами.

Функция init_board нужна для инициализации игровой доски, на которой в последствии будет соревнование 2 игроков, Основное поле 4 на 4 клетки в середине и созданы ячейки по краям, для того, что бы отслеживать ходы игроков за игровое поле, поэтому доска не 4 на 4, а 6 на 6.

```
def init_board():  
    board = np.zeros((6,6), dtype=int)  
    board[1][1] = 1 #p1  
    board[4][4] = -1 #p2  
    return board
```

Функция in_pl принимает значения 1 или -1 и нужна для инициализации 1 хромосомы (игрока), она содержит 4 основных значения типа double для определения дальнейших ходов, значение 1 или -1 для определения пола, значение для определения здоровья индивида и остальные для статистики по индивиду.

```
def in_pl(w=1): # w= 1 or -1? this gender  
    p1 = np.random.random(9)  
    p1[4] = w  
    p1[5] = 0  
    p1[6] = 0  
    p1[7] = 0  
    p1[8] = 0  
    return p1
```

Функция `init_players` инициализирует не 1 индивида, а 2, по такому же принципу, описанному выше.

```
def init_players():
    #f = open('population.txt', "r")
    p1 = np.random.random(6)
    p2 = np.random.random(6)
    p1[5] = 0
    p2[5] = 0
    p1[4] = 1
    p2[4] = -1
    return p1, p2
```

Функция `coord_check` нужна для возвращения координат игрока на доске функция соответственно принимает доску и пол (1 или -1), для этого использовались встроенные функции библиотеки `numpy`.

```
def coord_check(board, w):
    coord = []
    if w>0:
        coord.append(np.where(board>0)[0][0])
        coord.append(np.where(board>0)[1][0])
    elif w<0:
        coord.append(np.where(board<0)[0][0])
        coord.append(np.where(board<0)[1][0])
    return coord
```

Функция `coord_restruct` нужна для изменения координат игрока на доске, принимает соответственно доску, пол индивида и новые координаты

```
def coord_restruct(board, w, new_coord):
    try:
        x,y = coord_check(board, w)
        board[x][y] = 0
        x,y = new_coord
        board[x][y] = w
    except:
        print("Невозможно изменить координаты")
        return False
    return True
```

Функция `sigmoid` используется соответственно для вычисления сигмоиды. Нужна для перевода целых чисел в диапазон от 1 до 0. Смещение по оси X на 1.5 нужно для более точного определения ходов, так как они в диапазоне от 0 до 3..

```
def sigmoid(x):
    return 1/(1 + np.exp(-x+1.5)) # 0-1
```

Функция `finfunc` нужна для определения хода игрока, которое осуществляется в 4 направлениях соответственно числами 0, 1, 2, 3. Переводит число из диапазона 0-1 в целое.

```
def finfunc(x):
    if x > 0.75:
        x = 3
        return x
    elif x > 0.5 and x<=0.75:
        x = 2
        return x
    elif x > 0.25 and x<=0.5:
        x = 1
        return x
    elif x <=0.25:
        x = 0
        return x
```

Функция `move_math` является одной из основных функций генетического алгоритма, она принимает параметрами доску, игрока и предыдущие ходы двух игроков и на основе этих данных вычисляет дальнейших ход.

```
def move_math(board, p, last_move_me, last_move_p): # p - player, w = 1 or -1
    coord_me = coord_check(board, p[4])
    coord_p = coord_check(board, -p[4])
    f = (p[2]*(last_move_me/3) + p[3] * (last_move_p/3)) / (p[1]*math.sqrt(math.pow(coord_p[0] - coord_me[0],2) + math.pow(coord_p[1] - coord_me[1],2)) )
    # фитнес функция
    return finfunc(sigmoid(f)) # move
```

Функция `move_go` определяет новые координаты для игрока в зависимости от хода и перемещает игрока по доске, так же имеет возможность вернуть новые координаты не изменяя положения игрока на доске.

```
def move_go(board, move, w, cpu=False):
    x,y = coord_check(board,w)
    if move == 0:
        y-=1
    elif move == 1:
        x-=1
    elif move == 2:
        y+=1
    elif move == 3:
        x+=1
    if cpu == False: coord_restruct(board, w,[x,y])
    else: return [x,y]
```

Функция `check_move` проверяет новый ход игрока соответствию правилам игры, нельзя 2 раза ходить в одном направлении, пропуск хода при попытке выйти за границы доски, а также проверяет условие победы.

```
def check_move(board, w, move, last_move):
    if move == last_move:
        return 0 # game over
    x,y = coord_check(board, w)
    if x == 0 or x == 5 or y == 0 or y == 5:
        return 1 # skipping a move
    suuum = []
    for i in board:
        suuum.append(sum(i))
    if sum(suum) != 0:
        return 2 # winning
    return 'True'
```

Функция display выводит в консоль игровое поле.

```
def display(board):
    #a=1
    print("_"*90)
    print(board)
```

Функция cpu_move определяет ход и поведение однообразного компьютерного алгоритма, против которого и будет обучаться искусственный интеллект через генетический алгоритм.

```
def cpu_move(board, p, last_move):
    coord_p = coord_check( board, -p[4])
    move_pattern = [0,1,2,3]
    try:
        move_pattern.remove(last_move)
    except Exception:
        #print(Exception)
        a=1

    dist = []
    for i in move_pattern:
        coord_me = move_go(board,i,-p[4],True)
        dist.append([i, math.sqrt(math.pow(coord_p[0] - coord_me[0],2) + math.pow(coord_p[1] - coord_me[1],2))] )

    dist.sort(key=lambda dist: dist[1],reverse=True)
    return dist[0][0] # moove
```

Функция cross выполняет скрещивание хромосом родителей в случайном разрезе и отдает новую хромосому.

```
def cross(p1:list,p2:list):
    t = random.randint(1,3)
    p1[t:4]= p2[t:4].copy()
    return p1
```

Функция cri_ga является основной для генетического алгоритма, она определяет приспособленность индивида путем соревнования с компьютерным алгоритмом. В 5 ячейку хромосомы корректируется здоровье с учетом ходов индивида.

```

def cpu_ga(p):
    board = init_board()
    last_move_p, last_move_cpu = 4, 4
    move_p, move_cpu = None, None
    for o in range(50): # после 50 ходов в холостую идет ничья
        display(board)
        move_p = move_math(board, p, last_move_p, last_move_cpu)
        copy_board = board.copy()
        move_go(board, move_p, p[4])
        flag = check_move(board, p[4], move_p, last_move_p)
        if flag != 'True':
            if flag == 0:
                p[5] -= 3 # отнимаем здоровья за ход в 1 направлении
                p[6] += 1
                return p
            elif flag == 1:
                board = copy_board.copy()
                p[7] += 1
            elif flag == 2:
                p[8] += 1
                p[5] += 60
                print("player_win")
                return p
        p[5] += 2
        last_move_p = move_p
        move_cpu = cpu_move(board, p, last_move_cpu)
        display(board)
        move_go(board, move_cpu, -p[4])
        flag = check_move(board, -p[4], move_cpu, last_move_cpu)
        if flag != 'True':
            if flag == 0:
                p[6] += 5
                print("cpu_lag_move_x2")
                return p
            elif flag == 1: board = copy_board.copy()
            elif flag == 2:
                p[5] -= 60
                print("cpu_win")
                return p
        last_move_cpu = move_cpu
    print("прошло 50 ходов")
    p[5] -= 10
    return p

```

Функция `mut` соответственно выполняет мутацию случайного гена в хромосоме.

```

def myt(p: list):
    t = random.randint(1, 3)
    p[t] = abs(1 - p[t])
    return p

```

Функция `init_population64` соответственно создает популяцию индивидов в размере 64 штук и возвращает вложенный список с хромосомами. Так же функция имеет возможность создавать как разных чередующихся игроков (1 и -1) для соревнования между собой, так и всех полностью одинаковых (1) для соревнования с компьютерным алгоритмом.

```
def init_population64(cpu=False):
    population = []
    if cpu == False:
        for i in range(32):
            a,b = init_players()
            population.append(a)
            population.append(b)
        return population
    else:
        for i in range(64):
            population.append(in_pl(1))
        return population
```

Функция score аналогична сru_ga, однако здесь индивиды соревнуются между собой, а не с алгоритмом.

```
def score(p1,p2):
    board = init_board()
    last_move_p1,last_move_p2 = 5,5
    move_p1,move_p2,copy_board = None, None,None
    for o in range(50): # после 50 ходов в холостую идет ничья
        move_p1 = move_math(board,p1,last_move_p1, last_move_p2)
        copy_board = board.copy()
        move_go(board,move_p1, p1[4])
        display(board)
        flag = check_move(board,p1[4], move_p1, last_move_p1)
        if flag != 'True':
            if flag == 0:
                p1[5]-=2 # отнимаем здоровья за ход в 1 направлении
                p2[5]+=2
                print("Тех поражение")
                return p2, p1 # Тип техническое поражение
            elif flag == 1:
                board = copy_board.copy()
                print("Пропуск хода ")
            elif flag == 2:
                print("Выигрышь честным путем! ")
                p1+=10
                p2-=5
                return p1, p2
        p1[5]+=1
        last_move_p1 = move_p1
        copy_board = board.copy()
        move_p2 = move_math(board,p2,last_move_p2, last_move_p1)
        move_go(board, move_p2,p2[4])
        display(board)
        flag = check_move(board,p2[4], move_p2, last_move_p2)
        if flag != 'True':
            #print("Какого хера тут происходит ",flag )
            if flag == 0:
                p2[5]-=2
                p1[5]+=2
                print("Тех. поражение")
                return p1, p2 # Тип техническое поражение
            elif flag == 1:
                board = copy_board.copy()
                print("Пропуск хода ")
            elif flag == 2:
                p2+=10
                p1-=5
                print("Выигрышь честным путем! ")
                return p2, p1
        p2[5]+=1
        last_move_p2 = move_p2
        print(move_p1, move_p2)
    print("Ничья")
    return p1, p2
```


Функция `game_all_and_all64` принимает на вход популяцию из разных игроков и оценивает их приспособленностью путем соревнования между собой с помощью `score`, где каждый индивид соревнуется с каждым другим.

```
def game_all_and_all64(population:list): # каждый играет 64 раза (с каждым)
    for i in range(64):
        for j in range(i+1, 62):
            population[i],population[j] = score(population[i],population[j])
    return population
```

Функция `otbor_and_recombo` является так же основополагающей для генетического алгоритма, она отбирает лучших индивидов путем вероятностного отбора, который подразумевает, что чем больше у индивида здоровья, тем выше шанс пройти в следующее поколение. Далее производится скрещивание и мутация отобранных индивидов.

```
def otbor_and_recombo(population:list):
    population.sort(key=lambda population: population[5],reverse=True)
    new_population = []
    mi = min(abs(population[0][5]), abs(population[63][5]))
    ma = max(abs(population[0][5]), abs(population[63][5]))
    chance = abs(population[0][5])
    for individ in population:
        if abs(individ[5])/chance >= random.randint(mi,ma)/ma:
            new_population.append(individ) # отбираем новую популяцию вероятностным методом

    count = 64 - len(new_population)
    for i in range(0,count):
        new_population.append(cross(population[i],population[i+1]))

    for i in new_population:
        i[5]-=10
        if random.random() <= 0.5:
            i = myt(i)
    return new_population # может быть на 1 или несколько ячеек больше 64
```

Основной цикл алгоритма и отправная точка. Инициализируется популяция в 64 индивида со случайным набором хромосом для соревнования с ПК алгоритмом. Далее идет основной цикл поколений в котором во вложенном цикле перебираются и оцениваются приспособленность каждого индивида, далее идет отбор и рекомбинация поколения и цикл возвращается в начало к оценке приспособленности.

```
if __name__ == "__main__":
    population = init_population64(cpu = True)
    for o in range(1000):
        for i in range(len(population)):
            population[i] = cpu_ga(population[i])
        population = otbor_and_recombo(population)

    for i in population:
        print([round(j,2) for j in i])
```


Часть 3. Тестирование

В ходе проведения работы были проведены множественные тесты генетического алгоритма. На примере рисунка 1 можно увидеть эволюцию индивидов. Каждый список чисел — это отдельный индивид, первые 4 числа — это набор хромосом, которые задействованы для вычисления последовательности ходов в зависимости от ситуации в игре.

Пятое число означает пол индивида, здесь мы видим, что они все одинаковы. Шестое число — это здоровье конкретного индивида, чем оно больше, тем больше шанс попасть в следующее поколение.

Последние 3 числа — это статистика, первое из трех — кол-во поражений из-за хода в 1 направлении 2 раза подряд, второе — кол-во пропусков хода, третье — кол-во честных выигрышей.

```
[0.11, 0.9, 0.63, 1.0, 1.0, -804.0, 78.0, 130.0, 11.0]
[0.65, 0.46, 0.6, 0.35, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.49, 0.47, 0.52, 0.77, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.53, 0.52, 0.67, 0.56, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.24, 0.5, 0.73, 0.48, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.15, 0.43, 0.36, 0.45, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.34, 0.48, 0.47, 0.4, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.56, 0.48, 0.53, 0.33, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.8, 0.54, 0.29, 0.49, 1.0, -900.0, 100.0, 200.0, 0.0]
[0.51, 0.46, 0.61, 0.67, 1.0, -900.0, 100.0, 200.0, 0.0]
```

Рисунок 1 – Пример популяции 100 поколения

На рисунке 2 представлена однообразная популяция, скатившаяся в локальный минимум. Это произошло из-за неправильно выставленных вероятностей мутации и скрещивания, а также из-за некорректной оценки здоровья индивидов.

```
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
[0.51, 0.47, 0.19, 0.37, 1.0, -118725.0, 15903.0, 28087.0, 3.0]
```

Рисунок 2 – локальный минимум

ВЫВОД

Данный код реализует генетический алгоритм для простой игры, цель которой заключается в преследовании соперника на поле 4x4.

Разработано:

Функция создания популяции.

Функция приспособленности: вычисляет приспособленность индивидуума.

Функция отбора и рекомбинации: отбирает лучших представителей популяции в новый список, а затем производит скрещивание и мутацию индивидов с заданными вероятностями.

Навыки, полученные:

Использование генетического алгоритма для решения задачи оптимизации.

Работа с функцией приспособленности, операторами селекции, кроссовера и мутации.

Достоинства:

Гибкость и эффективность: Генетические алгоритмы обладают высокой гибкостью и могут применяться для решения различных задач оптимизации без необходимости знания аналитических решений.

Способность к обработке больших пространств поиска: Генетические алгоритмы могут работать с пространствами поиска большой размерности, что делает их полезными для сложных задач.

Недостатки:

Неопределенность: Результаты генетического алгоритма могут зависеть от параметров алгоритма, таких как размер популяции, вероятности кроссовера и мутации, качество разработки функции приспособленности, что требует тщательной настройки.

Вычислительная сложность: Для достижения хороших результатов могут потребоваться значительные вычислительные ресурсы, особенно при работе с большими пространствами поиска.

Улучшения:

Настройка параметров: Оптимизация параметров алгоритма может улучшить его производительность.

Адаптивные стратегии: Разработка адаптивных стратегий выбора параметров во время выполнения алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с. М.
2. Тим Джонс. Программирование искусственного интеллекта в приложениях / М. Тим Джонс ; Пер. с англ. Осипов А. И. - М.: ДМК Пресс, 2004. - 312 с: ил.
3. Вирсански Э. Генетические алгоритмы на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.: ил.
4. Голдберг Д. Генетические алгоритмы. / Голдберг Д. Г. - Издательство: ДМК Пресс, 2007. - 384 с. (ISBN: 978-5-94074-538-6)
5. Миттельманер В. Методы и модели оптимизации. Генетические алгоритмы и прочие методы глобальной оптимизации. / Миттельманер В. - Издательство: Физматлит, 2015. - 512 с. (ISBN: 978-5-9221-1624-8)
6. Хёльцль-Эрлер В. Генетическое программирование: проекты и задания: учебное пособие. / Хёльцль-Эрлер В. - Издательство: Питер, 2015. - 216 с. (ISBN: 978-5-4461-0179-2)
7. Миттельманер В. Генетические алгоритмы: принципы и практические применения. / Миттельманер В. - Журнал "Математика и информатика" №1, 2009, с. 89-97.
8. Дэвис Л., Голдберг Д., Рейнгольд Р. Генетические алгоритмы и искусственные нейронные сети. / Дэвис Л. и др. - М.: Техносфера, 2019. - 336 с. (ISBN: 978-5-94836-197-3)
9. Гросс Д. К. и др. Эксплуатация генетических алгоритмов: учебное пособие для вузов. / Гросс Д. К. и др. - Издательство: БХВ-Петербург, 2006. - 272 с. (ISBN: 5-94157-416-7)
10. Минд Л. Г. и др. Практикум по генетическим алгоритмам на Java. / Минд Л. Г. и др. - М.: БХВ-Петербург, 2012. - 320 с. (ISBN: 978-5-9775-1073-7)

11. Гладков, Леонид Генетические алгоритмы / Леонид Гладков. - Москва: Гостехиздат, 2020. - 868 с.
12. Хейнски И. и др. Методы и приложения генетического программирования. / Хейнски И. и др. - М.: Техносфера, 2018. - 352 с. (ISBN: 978-5-94836-369-4)
13. Холланд Д. Адаптивные методы поиска. / Холланд Д. - Санкт-Петербург: Питер, 2004. - 400 с. (ISBN: 5-94723-823-6)
14. Скобцов В.Ю. Интеллектуальный анализ данных: генетические алгоритмы: учеб.-метод. пособие / В.Ю. Скобцов, Н.В. Лапицкая, С.Н. Нестеренков. – Минск : БГУИР, 2018. – 92 с. : ил.
15. Мальцев, А.И. Алгоритмы и рекурсивные функции / А.И. Мальцев. - М.: [не указано], 2018. - 685 с.

ПРИЛОЖЕНИЕ 1

```
import numpy as np
import random
import math
# Данные которые принимает игрок:
# 1. свое положение в координатах
# 2. положение соперника в координатах
# 3. свой предыдущий ход (не может ходить в 1 направлении 2 раза
подряд)
# 4. предыдущий ход соперника (не может ходить в 1 направлении 2 раза
подряд)
# 5. сколько сделал ходов до победы
# 1.2 - возможно за место первых двух координат можно просто вычислять
расстояние до соперника,
# либо по обычной формуле для координат, либо как то с учетом
количества поворотов,
# мб расстояние - кол-во шагов кратчайшего пути до соперника на
текущей доске, однако это
# не дает информации в какой именно стороне противник
# 6. - heal

def save_p(population):
    f = open('population.txt', "w")
    for i in population:
        f.write(str(i)+ '\n')
    f.close()

def init_board():
    board = np.zeros((6,6), dtype=int)
    board[1][1] = 1 #p1
    board[4][4] = -1 #p2
    return board

def in_pl(w=1): # w= 1 or -1? this gender
    p1 = np.random.random(9)
    p1[4] = w
    p1[5] = 0
    p1[6] = 0
    p1[7] = 0
    p1[8] = 0
    return p1

def init_players():
    #f = open('population.txt', "r")
    p1 = np.random.random(6)
    p2 = np.random.random(6)
    p1[5] = 0
    p2[5] = 0
    p1[4] = 1
    p2[4] = -1
    return p1, p2

def coord_check(board, w):
    coord = []
    if w>0:
        coord.append(np.where(board>0) [0] [0])
        coord.append(np.where(board>0) [1] [0])
```

```

elif w<0:
    coord.append(np.where(board<0)[0][0])
    coord.append(np.where(board<0)[1][0])
return coord

def coord_restruct(board, w, new_coord):
    try:
        x,y = coord_check(board, w)
        board[x][y] = 0
        x,y = new_coord
        board[x][y] = w
    except:
        print("Невозможно изменить координаты")
        return False
    return True

def sigmoid(x):
    return 1/(1 + np.exp(-x+1.5)) # 0-1

def finfunc(x):
    if x > 0.75:
        x = 3
        return x
    elif x > 0.5 and x<=0.75:
        x = 2
        return x
    elif x > 0.25 and x<=0.5:
        x = 1
        return x
    elif x <=0.25:
        x = 0
        return x

def move_math(board, p, last_move_me, last_move_p): # p - player, w =
1 or -1
    coord_me = coord_check(board, p[4])
    coord_p = coord_check(board, -p[4])
    f =(p[2]*(last_move_me/3) +p[3]
*(last_move_p/3))/(p[1]*math.sqrt(math.pow(coord_p[0] - coord_me[0],2)
+ math.pow(coord_p[1] - coord_me[1],2)))
    # фитнес функция
    return finfunc(sigmoid(f)) # moove

def move_go(board, move, w, cpu=False):
    x,y = coord_check(board,w)
    if move == 0:
        y-=1
    elif move == 1:
        x-=1
    elif move == 2:
        y+=1
    elif move == 3:
        x+=1
    if cpu == False: coord_restruct(board, w, [x,y])
    else: return [x,y]

def check_move(board, w, move, last_move):
    if move == last_move:

```



```

        return 0 # game over
    x,y = coord_check(board, w)
    if x == 0 or x == 5 or y == 0 or y == 5:
        return 1 # skipping a move
    suuum = []
    for i in board:
        suuum.append(sum(i))
    if sum(suum) !=0:
        return 2 # winning
    return 'True'

def display(board):
    a=1
    #print("_"*90)
    #print(board)

def cpu_move(board, p, last_move):
    coord_p = coord_check( board, -p[4])
    move_pattern = [0,1,2,3]
    try:
        move_pattern.remove(last_move)
    except Exception:
        #print(Exception)
        a=1

    dist = []
    for i in move_pattern:
        coord_me = move_go(board,i,-p[4],True)
        dist.append([i, math.sqrt(math.pow(coord_p[0] - coord_me[0],2) +
math.pow(coord_p[1] - coord_me[1],2))])

    dist.sort(key=lambda dist: dist[1],reverse=True)
    return dist[0][0] # moove

def cpu_ga(p):
    board = init_board()
    last_move_p,last_move_cpu = 4,4
    move_p,move_cpu = None, None
    for o in range(50): # после 50 ходов в холостую идет ничья
        display(board)
        move_p = move_math(board,p,last_move_p, last_move_cpu)
        copy_board = board.copy()
        move_go(board, move_p, p[4])
        flag = check_move(board,p[4], move_p, last_move_p)
        if flag !='True':
            if flag == 0:
                p[5]-=3 # отнимаем здоровья за ход в 1 направлении
                p[6]+=1
                return p
            elif flag == 1:
                board = copy_board.copy()
                p[7]+=1
            elif flag == 2:
                p[8]+=1
                p[5]+=60
                print("player_win")
                return p
    p[5]+=2

```

```

last_move_p = move_p
move_cpu = cpu_move(board, p, last_move_cpu)
display(board)
move_go(board, move_cpu, -p[4])
flag = check_move(board, -p[4], move_cpu, last_move_cpu)
if flag != 'True':
    if flag == 0:
        p[6] += 5
        print("cpu_lag_move_x2")
        return p
    elif flag == 1: board = copy_board.copy()
    elif flag == 2:
        p[5] -= 60
        print("cpu_win")
        return p
    last_move_cpu = move_cpu
print("прошло 50 ходов")
p[5] -= 10
return p

def score(p1, p2):
    board = init_board()
    last_move_p1, last_move_p2 = 5, 5
    move_p1, move_p2, copy_board = None, None, None
    for o in range(50): # после 50 ходов в холостую идет ничья
        move_p1 = move_math(board, p1, last_move_p1, last_move_p2)
        copy_board = board.copy()
        move_go(board, move_p1, p1[4])
        display(board)
        flag = check_move(board, p1[4], move_p1, last_move_p1)
        if flag != 'True':
            if flag == 0:
                p1[5] -= 2 # отнимаем здоровья за ход в 1 направлении
                p2[5] += 2
                print("Тех поражение")
                return p2, p1 # Тип техническое поражение
            elif flag == 1:
                board = copy_board.copy()
                print("Пропуск хода ")
            elif flag == 2:
                print("Выигрышь честным путем! ")
                p1 += 10
                p2 -= 5
                return p1, p2
        p1[5] += 1
        last_move_p1 = move_p1
        copy_board = board.copy()
        move_p2 = move_math(board, p2, last_move_p2, last_move_p1)
        move_go(board, move_p2, p2[4])
        display(board)
        flag = check_move(board, p2[4], move_p2, last_move_p2)
        if flag != 'True':
            #print("Какого хера тут происходит ", flag)
            if flag == 0:
                p2[5] -= 2
                p1[5] += 2
                print("Тех. поражение")
                return p1, p2 # Тип техническое поражение

```

```

        elif flag == 1:
            board = copy_board.copy()
            print("Пропуск хода ")
        elif flag == 2:
            p2+=10
            p1-=5
            print("Выигрышь честным путем! ")
            return p2, p1
    p2[5]+=1
    last_move_p2 = move_p2
    print(move_p1, move_p2)
print("Ничья")
return p1, p2

def cross(p1:list,p2:list):
    t = random.randint(1,3)
    p1[t:4]= p2[t:4].copy()
    return p1

def myt(p:list):
    t = random.randint(1,3)
    p[t] = abs(1-p[t])
    return p

def init_population64(cpu=False):
    population = []
    if cpu == False:
        for i in range(32):
            a,b = init_players()
            population.append(a)
            population.append(b)
        return population
    else:
        for i in range(64):
            population.append(in_pl(1))
        return population

def game_all_and_all64(population:list): # каждый играет 64 раза (с
каждым)
    for i in range(64):
        for j in range(i+1, 62):
            population[i],population[j] = score(population[i],population[j])
    return population

def otbor_and_recombo(population:list):
    population.sort(key=lambda population: population[5],reverse=True)
    new_population = []
    mi = min(abs(population[0][5]), abs(population[63][5]))
    ma = max(abs(population[0][5]), abs(population[63][5]))
    chance = abs(population[0][5])
    for individ in population:
        if abs(individ[5])/chance >= random.randint(mi,ma)/ma:
            new_population.append(individ) # отбираем новую популяцию
    вероятностным методом

    count = 64 - len(new_population)
    for i in range(0,count):

```

```

        new_population.append(cross(population[i],population[i+1]))

for i in new_population:
    i[5]-=10
    if random.random() <= 0.4:
        i = myt(i)
return new_population # может быть на 1 или несколько ячеек больше
64

import time
if __name__ == "__main__":
    start_time = time.time()
    #population = init_population64()
    #for i in range(50):
    #    population = game_all_and_all64(population)
    #    population = othor_and_recombo(population)
    #print(population)
    population = init_population64(cpu = True)
    pop_copy = population.copy()
    for o in range(1000):
        for i in range(len(population)):
            population[i] = cpu_ga(population[i])
            population = othor_and_recombo(population)
    end_time = time.time()

    print(end_time - start_time, '\n'*4)

for i in population:
    print([round(j,2) for j in i])

```