" به نام یزدان پاک"

گزارش پروژه دوم

درس: مبانی هوش مصنوعی

استاد درس :استاد روشن فکر

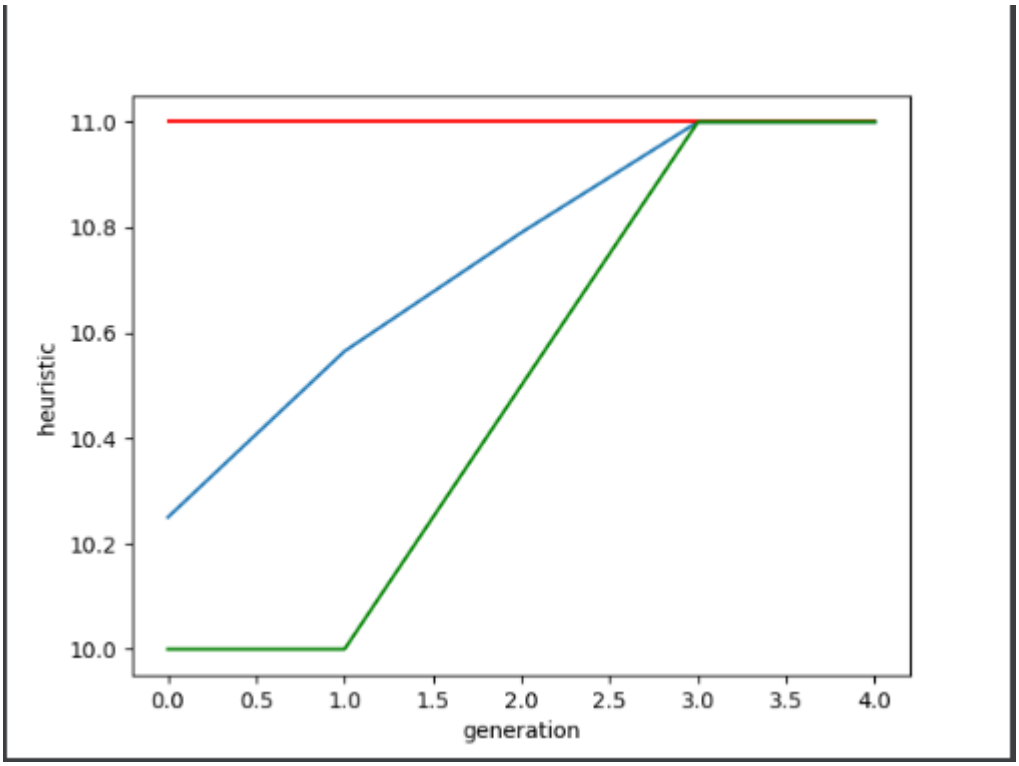
فرهان فرسی :9831094

محمد چوپان:9831125

در این پروژه هدف ما شبیه سازی بازی سوپر ماریو با استفاده از الگوریتم جهش است که در هر مرحله با تولید ژن ها به صورت تصادفی و استفاده از آن ها و ترکیب 200 ژن موجود در هر مرحله و در نهایت جهش ژن های به دست آمده الگوریتم که با توجه به تابع heuristic ما بیشترین امتیاز را کسب میکند را نمایش می دهیم.

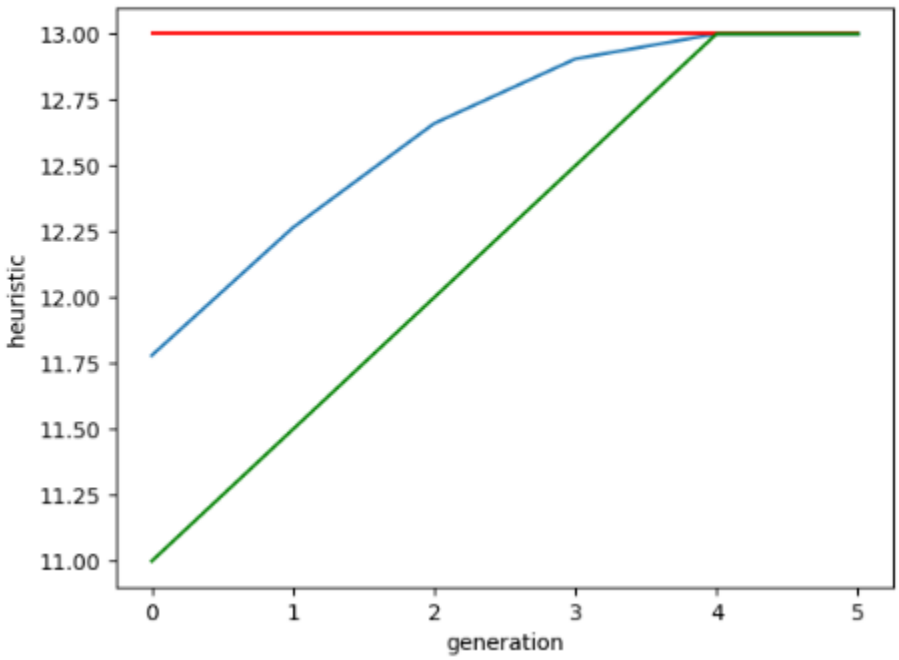
در هر مرحله نمودار مراحل طی شده را رسم میکنیم که نمودار آبی رنگ نمودار میانگین سبز بهترین حالت و قرمز بدترین حالت می باشد.

برای مرحله اول :

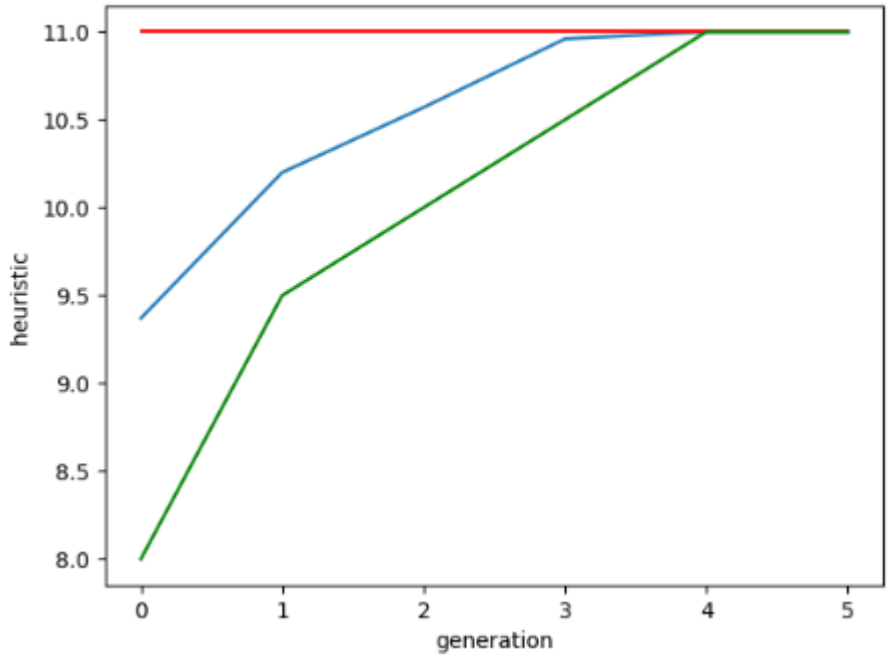


که الگوریتم ما در 4 نسل موفق به یافتن پاسخ مساله شده است.

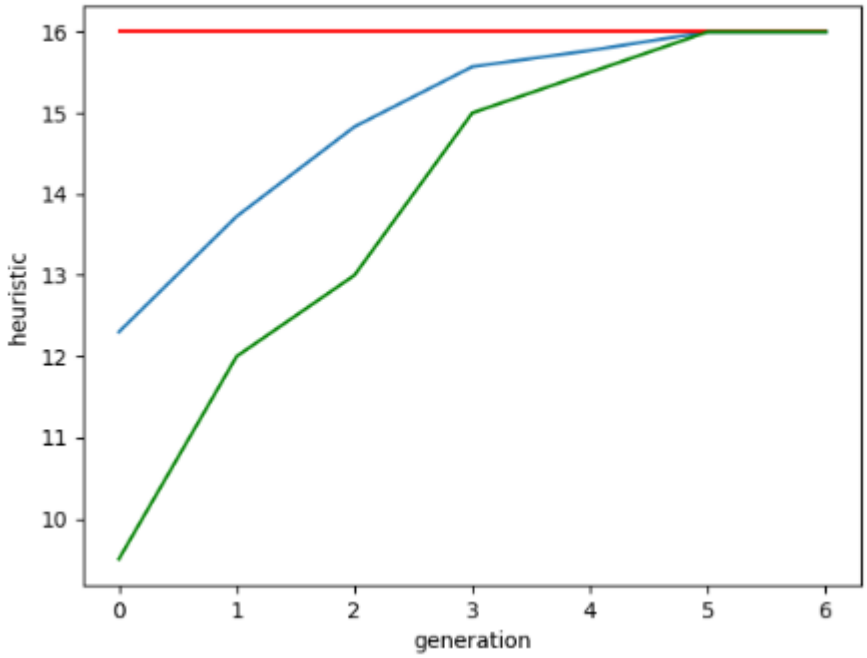
برای مرحله دوم:



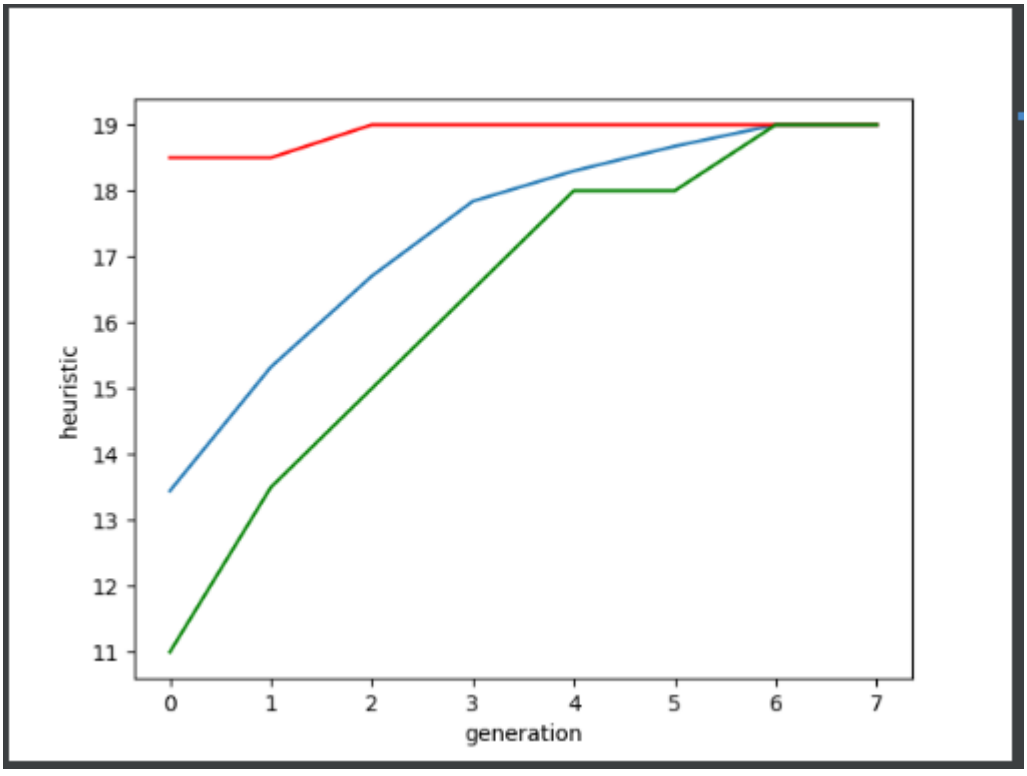
مرحله سوم:



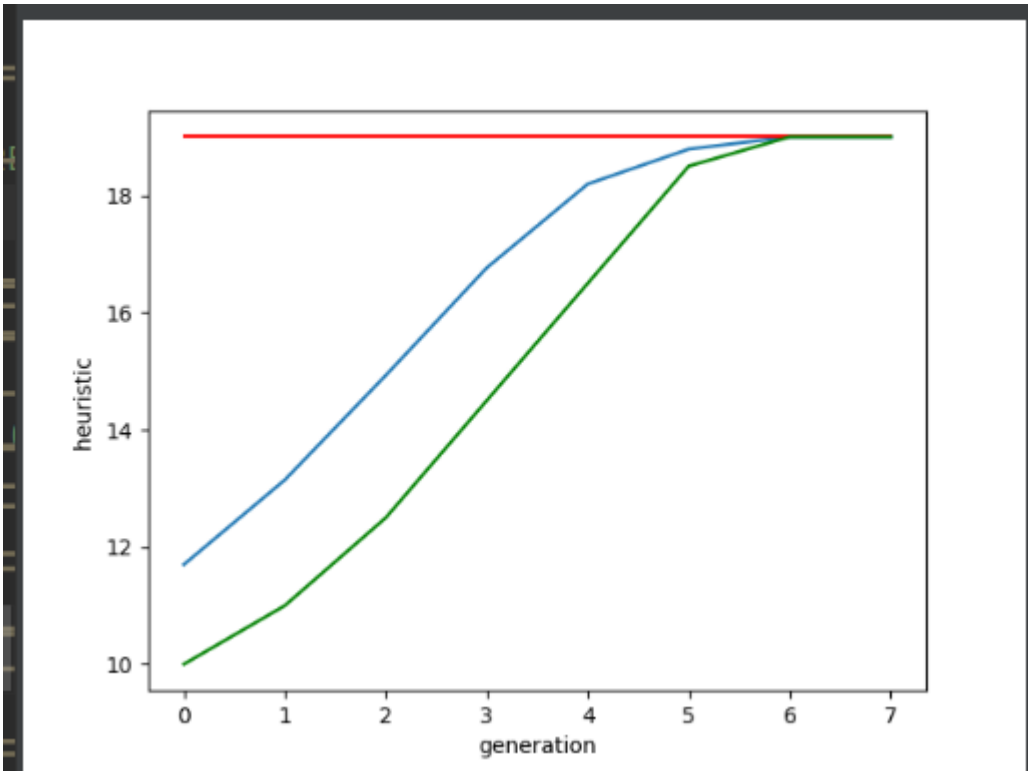
مرحله چهارم:



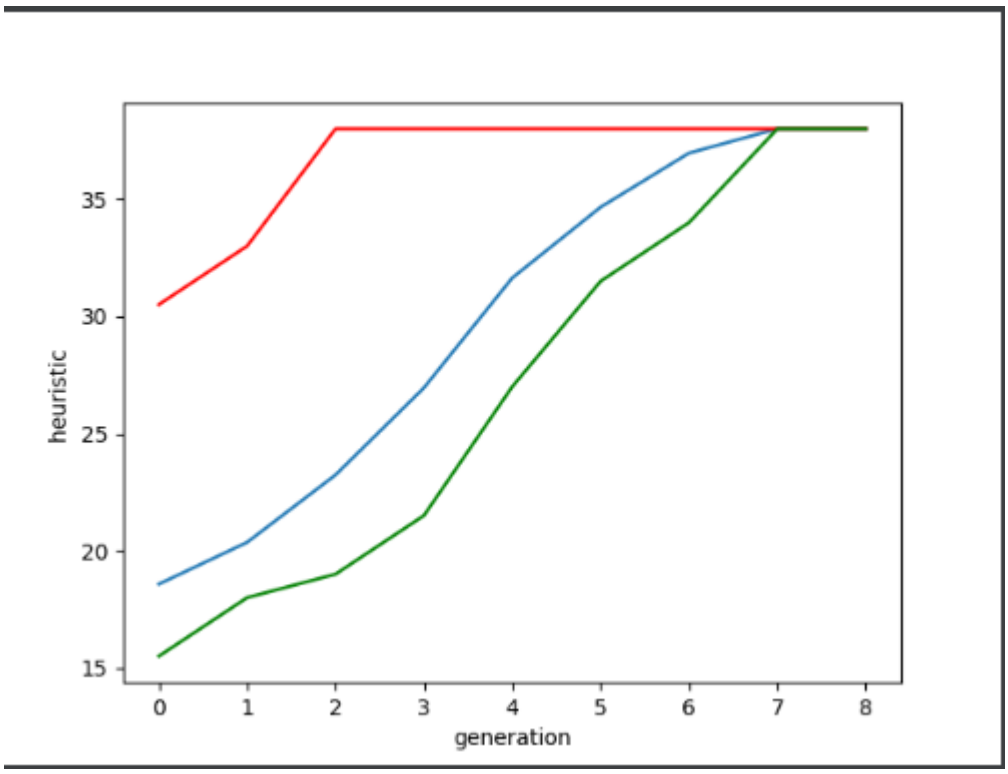
مرحله پنجم:



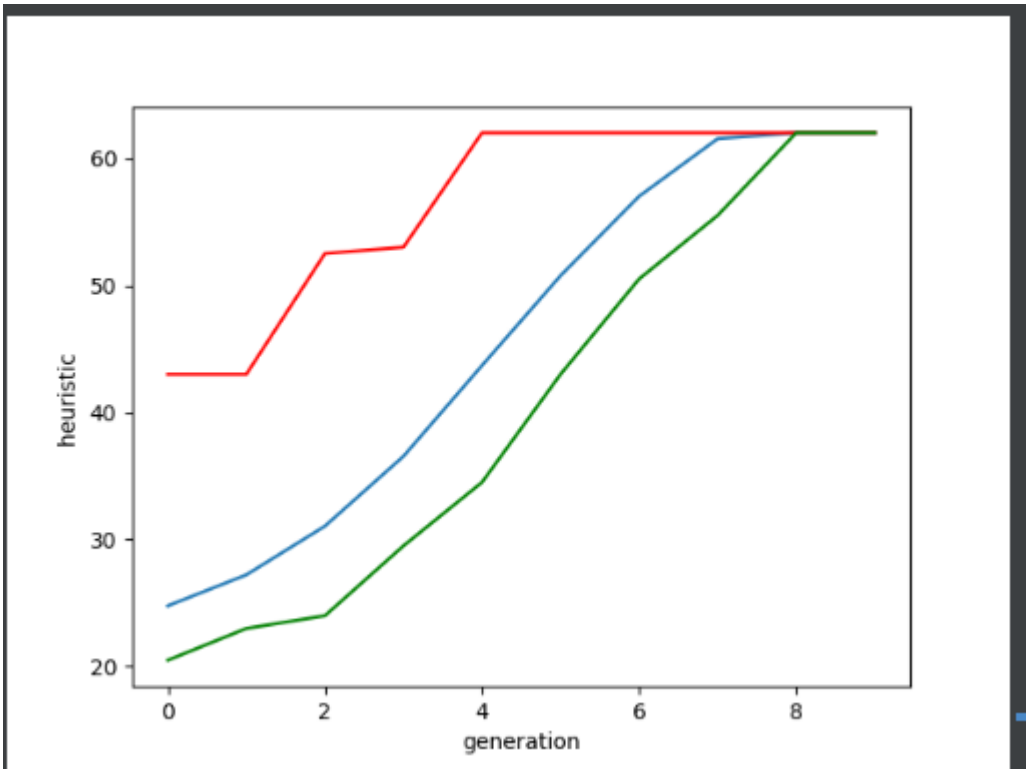
ششم:


heuristic
generation
10
12
14
16
18
0
1
2
3
4
5
6
7

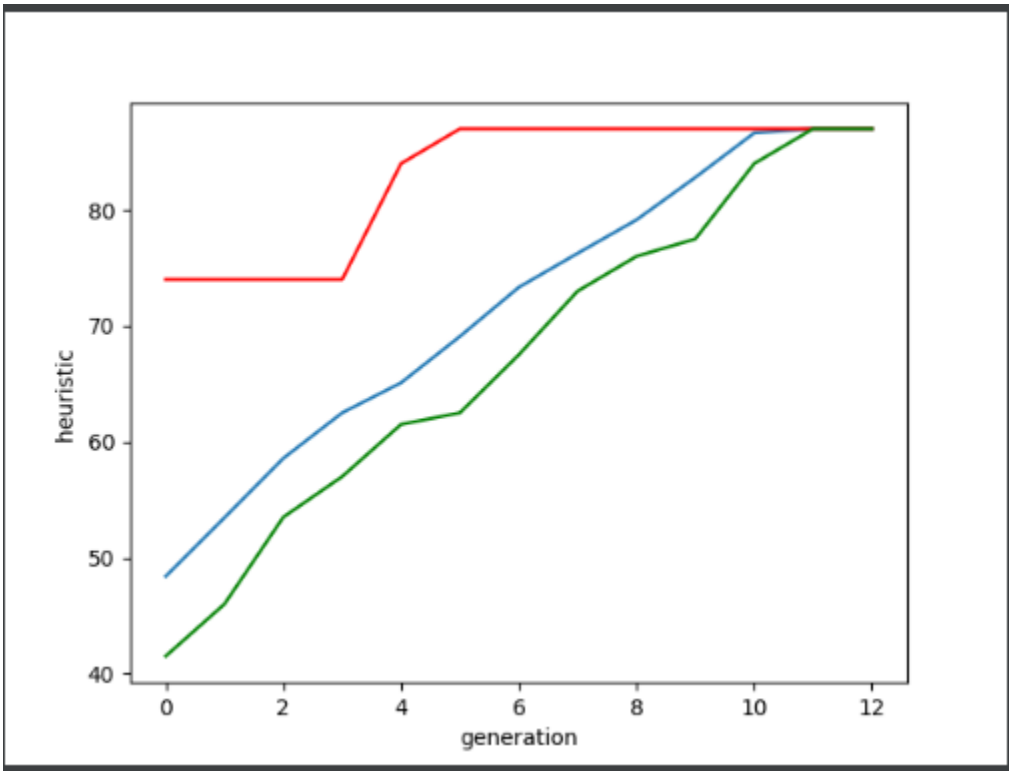
هفتم:


heuristic
generation
15
20
25
30
35
0
1
2
3
4
5
6
7
8

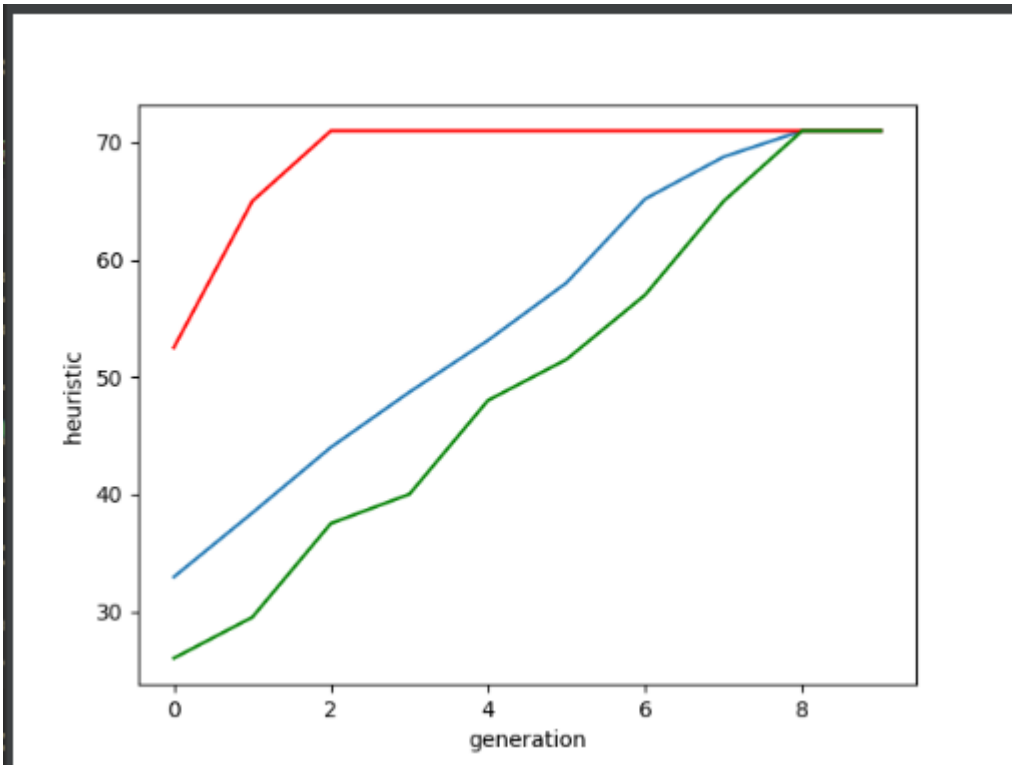
هشتم:


heuristic
generation
20
30
40
50
60
0
2
4
6
8

نهم:


heuristic
generation
40
50
60
70
80
0
2
4
6
8
10
12

دهم:



در کل با مقایسه نمودار های به دست آمده از شبیه سازی ما متوجه می شویم که :

برا ی جمعی ت او لیه تعداد کروموزوم بیش تر یعنی نمونه اولیه ب بیش تر متیو اند باعث شود جواب بهینه تری به دست بیا وریم.
برای محاسبه heuristic در نظر گرفتن امتیاز برنده شدن تابع بهتری برای پیروزی داریم.
ولی بهتر است امتیاز زیادی در نظر گر فته نشود که هدف فقط برنده شدن نباشد.
برا ی انتخاب وزن دار بودن حرکات بر اسا س شایستگی میتو اند گزینه بهتری باشد.
برا ی با زتکریبی دو نقطه بر ا ی رسیدن به جواب بهتر میتواند گزینه خوبی باشد.
برای جهش احتمال جهش 0.1 می تواند به سرعت همگرایی بیشتری ختم شود.

کد ها :

در این کلاس هر یک ژن مجموعه از آن ها و تابعی برای ساخت آن ها تعریف می کنیم.

```
class Chromosome:
    def __init__(self, length):
        self.chromosome_cell = []
        self.chromosomes = []
        self.n = length

    '''
    build each chromosome randomly
    '''

    def build_chromosome(self):
        self.chromosome_cell = []
        choice = [0, 2]
        for i in range(self.n):
            temp = random.randint(0, 2)
            if i > 0 and self.chromosome_cell[i - 1] == 1 and temp == 1:
                temp = random.choice(choice)
            self.chromosome_cell.append(temp)
        return self.chromosome_cell
```

این تابع برای این است که بهه تعداد 200 مرتبه در ابتدای کار برای ما ژن تولید کند.

```
def build_all_chromosomes(self):
    for i in range(200):
        each_chromosome = self.build_chromosome()
        self.chromosomes.append(each_chromosome)
    return self.chromosomes
```

تعریف کلاس heuristicبرای کل و تعریف مرحله های بازی در آن

```python
class Heuristic:
    def __init__(self, levels):
        self.levels = levels
        self.current_level_index = -1
        self.current_level_len = 0

    '''
    load the next level of game
    kind of unused:/
    '''

    def load_next_level(self):
        self.current_level_index += 1
        self.current_level_len = len(self.levels[self.current_level_index])
```

تابعی برای امتیاز دادن به هر کروموزوم با توجه به آن مرحله از بازی

```python
def get_score(self, actions):
    current_level = self.levels[self.current_level_index]
    steps, max = 0, 0
    win = True
    for i in range(self.current_level_len):
        current_step = current_level[i]
        if current_step == '_':
            steps += 1
            if i > 1 and actions[i - 2] == '1':
                steps -= 0.5
        elif current_step == 'M':
            if (i > 0 and actions[i - 1] != '1') or i == 0:
                steps += 3
            if i > 1 and actions[i - 2] == "1":
                steps -= 0.5
        elif current_step == 'G' and actions[i - 2] == '1' and i > 1:
            steps += 3
        elif current_step == 'G' and actions[i - 1] == '1':
            steps += 1
        elif current_step == 'L' and actions[i - 1] == '2':
            steps += 1
        else:
            steps = 0
            win = False
        if max < steps:
            max = steps
    if steps == self.current_level_len:
        max += 5
    if actions[len(actions) - 1] == '1':
```

تعریف کلاس اصلی بازی که در فایل پیوست پروژه وجود داشت و تابعی برای محاسبه امتیاز تمامی ژن های تولید شده برای یک مرحله از بازی

```
class Game:
    def __init__(self, heuristic, chromosome):
        self.heuristic = heuristic
        self.chromosome = chromosome
        self.chromosomes = chromosome.build_all_chromosomes()
        self.average = 0
        self.avg_plot = []
        self.best = []
        self.worst = []

    '''
    calculate the score of all chromosomes of each generation
    '''

    def score_all(self):
        population = []
        for i in range(len(self.chromosomes)):
            string_chromosome = ''
            for j in range(len(self.chromosomes[i])):
                string_chromosome += str(self.chromosomes[i][j])
            population.append([string_chromosome, self.heuristic.get_score(string_chromosome)])
        population = self.sort(population)
        return population
```

انتخاب ژن های برتر بین ژن های تولید شده

```
def choose(self, children):
    grandchildren = []
    for i in range(int(len(children) / 2)):
        child1, child2 = self.cross_over(children)
        grandchildren.append(child1)
        grandchildren.append(child2)
    children = self.next_generation(children, grandchildren)
    children = self.mutation(children)
    self.game_over(children)
    return children
```

تولید ژن های جدید با توجه به ژن های برتر گزینش شده

```
def cross_over(self, children):
    random1 = random.randint(0, len(children) - 1)
    random2 = random.randint(0, len(children) - 1)
    chromosome1 = list(children[random1][0])
    chromosome2 = list(children[random2][0])
    for i in range(int(len(children[0][0]) / 2), len(children[0][0])):
        chromosome1[i], chromosome2[i] = chromosome2[i], chromosome1[i]
    chromosome1 = ''.join(chromosome1)
    chromosome2 = ''.join(chromosome2)
    return [chromosome1, h.get_score(chromosome1)], [chromosome2, h.get_score(chromosome2)]
```

جهش ژن های گزینش شده

```
def mutation(self, grandchildren):
    k = random.randint(0, len(grandchildren[0][0]))  # change how many cells
    for j in range(len(grandchildren)):
        for i in range(k):
            yes_no = random.choices([0, 1], weights=(80, 20), k=1)  # yes or no
            if yes_no == 1:
                change = random.randint(0, len(grandchildren[0][0]))
                grandchildren[j][0][change] = 0  # reset to zero
    return grandchildren
```

تولید نسل بعدی

```
def next_generation(self, children, grandchildren):
    for i in range(len(children)):
        grandchildren.append(children[i])
    grandchildren = self.sort(grandchildren)
    children = []
    for i in range(int(len(grandchildren) / 2)):
        children.append(grandchildren[i])
    return children
```

رسم نمودار تابع heuristic :

```
def heuristic_avg(self, grandchildren):
    sum_value = 0
    for i in range(len(grandchildren)):
        sum_value += grandchildren[i][1][0]
    average = sum_value / len(grandchildren)
    self.avg_plot.append(average)
    self.best.append(grandchildren[0][1][0])
    self.worst.append(grandchildren[99][1][0])
    return average
```

تابع اصلی برنامه خواندن از فایل و بارگزاری هر مرحله و محاسبات آن:

```
if __name__ == '__main__':
    num = 1
    for i in range(10):
        file_name = "level" + str(num) + ".txt"
        f = open(file_name, "r")
        content = f.read()
        print(content)
        h = Heuristic([content])
        h.load_next_level()
        Chro = Chromosome(len(content))
        game = Game(h, Chro)
        population = game.score_all()
        game.build_children(population, [])
        num += 1
```