מעבדה בבינה מלאכותית

חלק ב

סעיף 1:

פונקציית אתחול ה TARGET

```
def init_target(self):
    for i in range(self.target_size):
        self.target += str(random.randrange(0, 2))
    print("Target --> ", self.target)
```

פונקציית אתחול שנייה

```
def init_str(self, target_size):
    for i in range(target_size):
        rand = random.randrange(0, 4)
        if rand == 0 or rand == 1:
            self.str += '?'
        if rand == 2:
            self.str += '0'
        if rand == 3:
            self.str += '1'
```

פונקציית SEARCH_LOCAL

```
def local_search(self):
    self.correct = 0
    self.incorrect = 0
    for i in range(self.iter):
        for j in range(self.popsize):
            if not self.population[j].success:
                new_str = ""
                for k in range(len(self.population[i].str)):
                    c = self.population[j].str[k]
                    if c == '?':
                        new_str += str(random.randrange(0, 2))
                    else:
                        new_str += c
                    if i == 0 and c != '?':
                        if c == self.target[k]:
                            self.correct += 1
                        else:
                            self.incorrect += 1

                if new_str == self.target:
                    self.population[j].success = True
                    self.population[j].attempts_num = i + 1
```

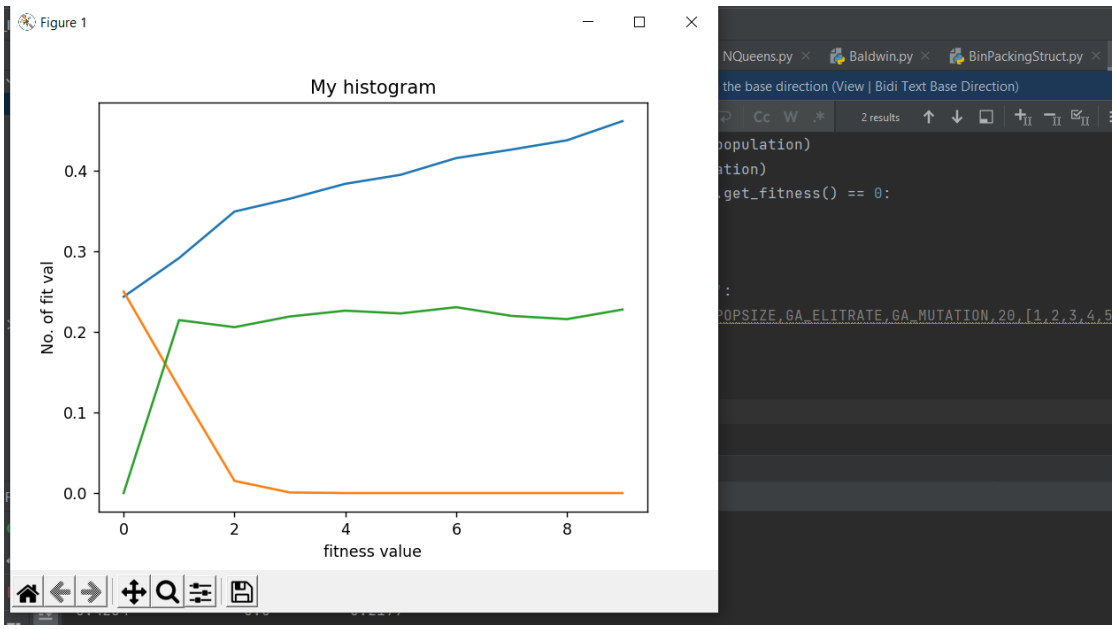
פונקציית FITNESS_CALC

```
def calc_fitness(self):
    self.fitness_sum = 0
    for i in range(self.popsize):
        n = self.iter - self.population[i].attempts_num
        if not self.population[i].success:
            n = 0
        self.population[i].fitness = 1 + (19 * n / 1000)
        self.fitness_sum += self.population[i].fitness
        #if self.population[i].success:
            #print(i, "- Fitness: ", self.population[i].fitness, "Attempts: ", self.population[i].attempts_num)

        self.population[i].success = False
```

פונקציית MATE כדי לעסות בנים

```
def mate(self):
    prob_arr = []
    self.learnedbits =0
    for i in range(self.popsize):
        prob_arr.append(self.population[i].fitness / self.fitness_sum)
    for i in range(self.popsize):
        i1 = numpy.random.choice(self.population, p=prob_arr)
        i2 = numpy.random.choice(self.population, p=prob_arr)
        myStr = ''
        for j in range(len(self.target)):
            spos = random.randint(0, 1)
            if spos == 0:
                myStr += i1.str[j]
                if self.population[i].str[j]=='?' and i1.str[j]!='?':
                    self.learnedbits+=1
            else:
                myStr += i2.str[j]
                if self.population[i].str[j] == '?' and i2.str[j] != '?':
                    self.learnedbits += 1

        self.buffer[i].str = myStr
    self.population = self.buffer
```

ואז הוספנו משתנים כדי לחשב CORRECT POSSITION AND WRONG POSSITION מצאנו שכמו שציפינו שCORRECT POSSITION עולה אבל ה WRONG יורד שזה טבעי כי אנחנו עושים MATE לאיברים הכי טובים בהסתברות גדולה כמו שרואים בגרף הזה



קוו כחול זה CORRECT POS ואדום זה WRONG ירוק LEARNED BITS

סעיף 2:

הוספנו את הפרמטרים כנדרש

ואז הקוד של HILL CLIMBING

```
def hill_climb(self, state):
    if state.fitness == 0:
        return state
    nearstates = self.calc_near_states(state)
    for city in nearstates:
        if city.fitness < state.fitness:
            state = city
            return state
        if state.fitness == 0:
            return state
    return state
```

דוגמא לריצה עבור בול פגיע

```
C:\Users\win10\AppData\Local\Programs\Python\Python310\python.exe C:/Users/win10/PycharmProjects/AI_LAB1/main.py
Best:  Yd]kl3XosrFL  ( 137 )
Best:  Yd]kl3XosrjL  ( 113 )
Best:  Ed]kl3XosrjL  ( 99 )
Best:  Edykl3XosrjL  ( 97 )
Best:  Edykl3XosgjL  ( 96 )
Best:  Edykl3Xosgj7  ( 75 )
Best:  Edykl3Xosgj7  ( 75 )
Best:  Efjim*_ii]f(  ( 68 )
Best:  J`mtn-\rzgc0  ( 67 )
Best:  J`mtn-Wrzgc0  ( 62 )
Best:  Hgomm cjurj$  ( 43 )
Best:  Hgomm cjurd$  ( 37 )
Best:  Hgkmm cjurd$  ( 35 )
Best:  Hajku$Xrtkg#  ( 29 )
Best:  Hajku"Xrtkg#  ( 27 )
Best:  Hajku"Wrtkg#  ( 26 )
Best:  Ifkkl!Vrqpf#  ( 21 )
Best:  Ifkkl!Vrqpf#  ( 21 )
Best:  Ifkkl!Vrqpf#  ( 21 )
Best:  Ifkkl!Vrqpf#  ( 21 )
Best:  Ifnko!Pnsjc   ( 19 )
Best:  Jgjio Xovkd!  ( 15 )
Best:  Jejio Xovkd!  ( 13 )
Best:  Jejio Xotkd!  ( 11 )
Best:  Jejio Xotkd!  ( 11 )
```

```
Best:  Helko Wprld!  ( 2 )
Best:  Helko Wprld!  ( 2 )
Best:  Helko Wprld!  ( 2 )
Best:  Helko Wprld!  ( 2 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello!World!  ( 1 )
Best:  Hello World!  ( 0 )

Process finished with exit code 0
```

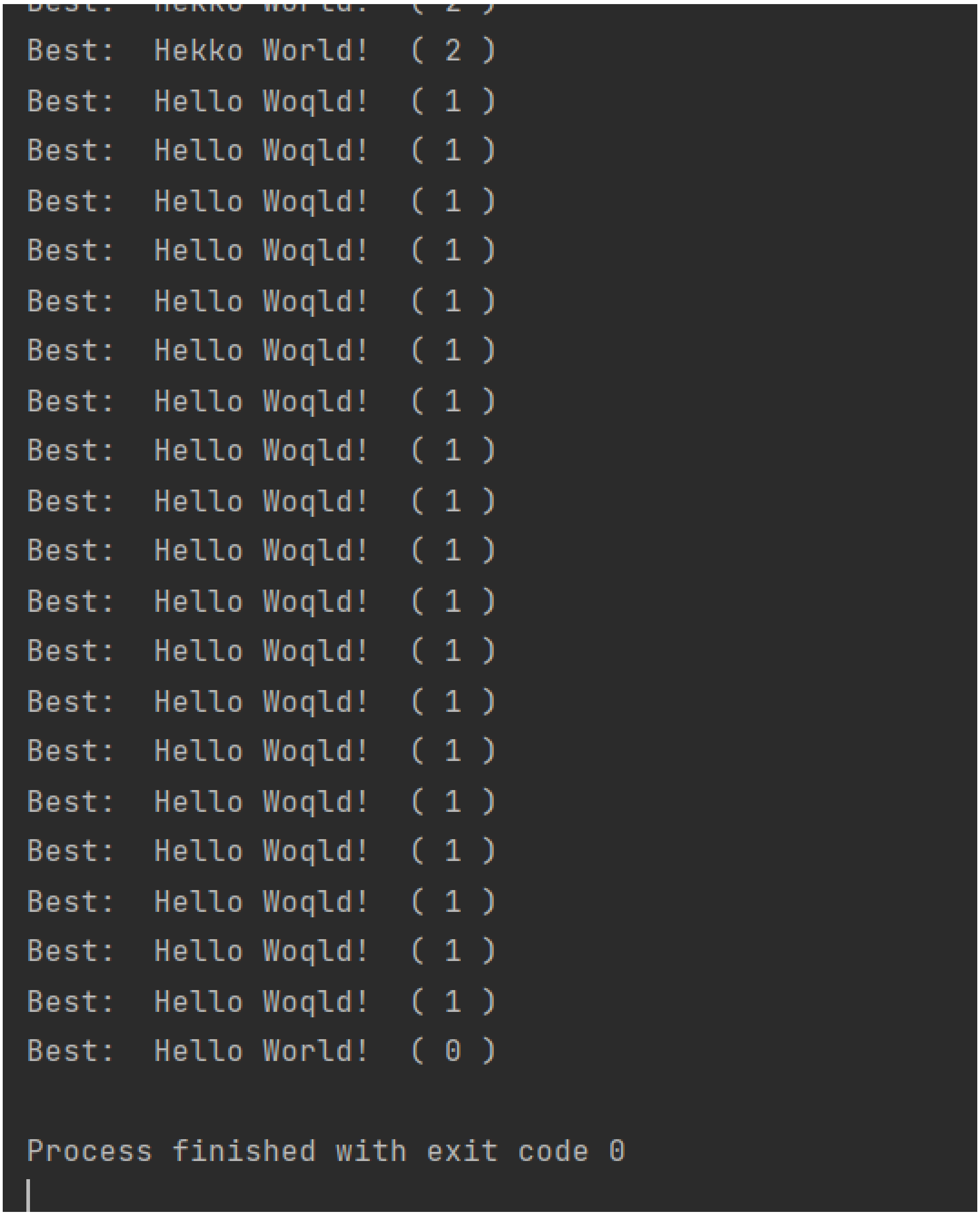
אז מימשנו STEEPEST ASCEND

שבוחר הבן המשפר ביותר ולא הראשון

```python
def steepest_ascend(self, state):
    if state.fitness == 0:
        return state
    nearstates = self.calc_near_states(state)
    for city in nearstates:
        if city.fitness < state.fitness:
            state = city
        if state.fitness == 0:
            return state
    return state
```

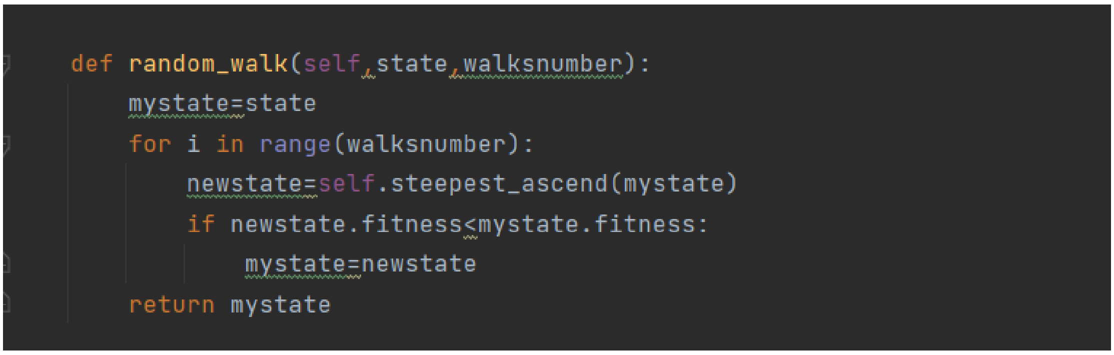
דוגמת ריצה:

```
Best:  BUv]a.Yhstm:  ( 127 )
Best:  DWbqq<\\zed$  ( 105 )
Best:  DWbqq3\\zed$  ( 96 )
Best:  I\mqu:S^yjc"  ( 80 )
Best:  I^ton!Yesbh+  ( 58 )
Best:  I^ton!Yesbh+  ( 58 )
Best:  I^ton!Yesbc+  ( 55 )
Best:  Iejxi#\frsd!  ( 45 )
Best:  I^ton!Ypsbc#  ( 38 )
Best:  Iejri#\mrsd!  ( 32 )
Best:  Ielri#\mrsd!  ( 30 )
Best:  Ielli#\mrsd!  ( 24 )
Best:  Ielli#Zmrsd!  ( 22 )
Best:  Hcnpo Sowie!  ( 21 )
Best:  Hcnpo Sowie!  ( 21 )
Best:  Hcnpo Sowie!  ( 21 )
Best:  Idkjn!Xoshc#  ( 16 )
Best:  Idkjn!Xoshc#  ( 16 )
```



```
Best:  Hekko World!  ( 2 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello Woqld!  ( 1 )
Best:  Hello World!  ( 0 )

Process finished with exit code 0
```

ואז מימשנו את RANDOM WALK



```
def random_walk(self,state,walksnumber):
    mystate=state
    for i in range(walksnumber):
        newstate=self.steepest_ascend(mystate)
        if newstate.fitness<mystate.fitness:
            mystate=newstate
    return mystate
```

דוגמת ריצה : שזה הכי טוב שהיה מבין שלושתם :

```
C:\Users\win10\AppData\Local\Programs\Python\Python310\python.exe C:/Users/win10/PycharmProjects/AI_LAB1/main.py
Best:  Kdmkq Woqke   ( 12 )
Best:  Henlo Workd!  ( 3 )
Best:  Henlo World!  ( 2 )
Best:  Hello World!  ( 0 )

Process finished with exit code 0
```

```
def calc_near_states(self, state):
    nearstates = []
    for i in range(self.N):
        for j in range(i):
            newArr = []
            for k in range(self.N):
                newArr.append(state.NQueens[k])
            newstate = GAStruct(self.N, newArr)
            tmp = newstate.NQueens[i]
            newstate.NQueens[i] = newstate.NQueens[j]
            newstate.NQueens[j] = tmp
            nearstates.append(newstate)
    self.calc_fitness_states(nearstates)
    return nearstates
```

K GENES עבור בעיית בול פגיעה:

```
def k_gene_exchange(population):

    child = GAStruct("",10)
    for i in range(len(GA_TARGET)):
        child.str += population[randint(0, GA_POPSIZE / 2)].str[i]
    return child
```

K GENES עבור בעיית NQUEENS

```
def k_gene_exchange(self):
    arr=[]
    for i in range(self.N):
        arr.append(0)
    child = GAStruct(self.N,arr )
    for i in range(self.N):
        child.NQueens[i] = self.population[randint(0, self.GA_POPSIZE / 2)].NQueens[i]
    return child
```

K GENES עבור בעיית BINPACKING

```python
def k_gene_exchange(self):

    child = BinPackingStruct()
    child.updateN(self.N)
    for i in range(self.N):
        child.arr[i] = self.population[randint(0, self.GA_POPSIZE / 2)].arr[i]
```

סעיף אחרון השוואת אלגוריתמים :

א) כל האלגוריתמים בשתי המעבדות הם שלימים מוצאים פתרון
ב) איצולינו עבור בעיית בול פגיעה ו NQUEENS במעבדה הזו תמיד מוצא פתרון אופטימאלי וגם במעבדה הישנה אם משתמשים ב MINIMAL CONFLICTS עבור NQUEENS בסוף מגיעים לתוצאה הנדרשת אבל ב BINPACKING לא תמיד מוצא את האופטימלי אבל במעבדה הזו הגיע לפתרונות יותר טובים
ג) במעבדה הזו מהירות ההתכנסות מאוד גבוהה בשלושת האלגוריתמים במעבדה הישנה הייתה טובה עבור בעיית ה NQUEENS ובול פגיעה אבל BINPACKING לפעמים דרש יותר זמן
ד) זמן ריצה במעבדה הישנה היה מאוד מהיר למשל MINIMAL CONFLICT רץ מאוד מהיר וגם במעבדה הזו כולם היו לא איטים בגלל שכל איטירציה דרשה יותר זמן אבל היה פחות איטירציות