AI Lab 4 part 2

CoEvolution

Submitted BY :

Abed Abo Hussien 208517631

Samer Kharaoba 209050202

First of all the changes done to the algorithm are as follows:

1. Mutualism function
2. Communalism function
3. Parasitism function

We used all three to create the new algorithm, by replacing the cross function in the previous GA with all three stages.

1. Each stage takes a gene and manipulates every single item in that gene:
2. Each gene from the population is comprised of a string $\in \{0,1,2\}$ with length 1000
3. The implementation: we took each item in each string as a vector X and used the equations that we learned in the lecture to change them
   - For example :
   - $X_{best} = [0,0,1], X_i = [1,0,2]\ , X_j = [2,0,1] \rightarrow we\ took\ X_{best}[1], X_i[1], X_j[1]$
   - Then applied each equation on each set

Now lets explain each stage :

Mutualism function:

- It was implemented as described in the lecture and we already explained how we view each "vector" as stated above
- Select a random X_j ,get random Benefit factor $\in \{1,2\}$,Create Mutual vectors ,use all of the above in the equation
- Then update the $Organisim_i\ and\ Organism_j\ (X_i, X_j) according\ to\ the\ fitness\ function$

```
def mutualism(self):
    print("------Mutualism phase-----------")
    for i in range(len(self.population)):
        organizm, organizm_i, organizm_j = self.select_XJ(i)
        BF = [random.randint(1, 2), random.randint(1, 2)]
        Mutual_vectors = [(x_i + x_j) // 2 for x_i, x_j in zip(organizm_i, organizm_j)]
        organizmz = [organizm_i, organizm_j]
        best = self.solution.object
        index_to_update = [i, organizm]
        for org in range(2):
            zipped = zip(organizmz[org], Mutual_vectors)
            item = [(x_i + int(random.uniform(0, 1) * abs(best[index] - mv * BF[org]))) % 3 for index, (x_i, mv) in
                    enumerate(zipped)]
            self.Update_Org_fit(index_to_update[org], item)
```

Communalism function

- Exactly as above function but now update only $Organism_i$

```
def communalism(self, esize, birth_count):
    print("------Communalism phase----------")

    for i in range(esize, esize + birth_count):
        organizm, organizm_i, organizm_j = self.select_XJ(i)
        best = self.solution.object
        zipped = zip(organizm_i, organizm_j, best)
        organizm_i_new = [x_i + random.uniform(-1, 1) * abs(x_best - x_j) for x_i, x_j, x_best in zipped]
        organizm_i_new = [int(xi) % 3 for xi in organizm_i_new]
        self.Update_Org_fit(i, organizm_i_new)
```

- The thing to note about Communalism and Mutualism is that they replace the cross functionality

Parasitism

- **The nice thing about this function Is that it mutates a section of the population , and replaces the mutate functionality , but it is more aggressive than the mutation function of the original GA.**
- **Exactly the same parameters as the previous functions but adds the mutation probability**
- **Parasite vector is calculated based on the mutation probability**
- **We can use adaptive decrees to change the ratio of exploration vs exploitation**

```
def parasitism(self, esize, birth_count, gen):
    print("------Parasitism faze-----------")
    for i in range(esize, esize + birth_count):
        mutation = GA_MUTATION if (
                (not self.hyper_mutaion) and self.trigger_mutation) \
            else maxsize * adaptive_decrease(0.75, 1, gen)
        organizm, organizm_i, organizm_j = self.select_XJ(i)
        Parasite_vector = [organizm_i[index] if random.uniform(0, 1) >= mutation else
                           random.randint(0, 2) for index in range(len(organizm_i))]
        self.Update_Org_fit(i, Parasite_vector)
```

We also created a match between our agent and all dummies and experts as a guideline for the fitness:

1. For every citizen ,check how many matches it **<u>doesn't lose</u>** and we put that as the fitness
2. Granted this fitness changes from match to match, but it also means that we get competitive benchmarks in between the population
3. And a function called winner that determines the winner of the game

```
def rashembo_match(self, civilian, agent):

    score = 0
    rps_to_text = ('rock', 'paper', 'scissors')
    for i in range(1000):
        next = agent.nextMove()
        score += self.Winner(civilian.object[i], next)
        agent.storeMove(civilian.object[i], score)

    return score, score >= 0

def rashembo_match2(self, civilian, agent):
    rps_to_text = ('rock', 'paper', 'scissors')
    rps_to_num = {'rock': 0, 'paper': 1, 'scissors': 2}
    score = 0
    greenberg_moves = []
    my_moves = [rps_to_text[civilian.object[i]] for i in range(len(civilian.object))]
    for i in range(1000):
        move = player(my_moves[:i], greenberg_moves)
        score += self.Winner(civilian.object[i], rps_to_num[move])
        greenberg_moves.append(move)
    return score, score >= 0

def rashembo_match3(self, civilian, agent):
    score = 0
    for i in range(1000):
        move = iocaine_agent(i, civilian.object[i - 1])
        score += self.Winner(civilian.object[i], move)
    return score, score >= 0
```

The function called fit_dummies established 1000 matches between our agent and the rest of the robots and calculates the fitness as the number of wins against all robots divided by the number of robots :

```
def fit_dummies(self, population, dummies):

    for index, pop in enumerate(population):
        wins = 0
        robots = []
        self.init_dummies()
        for dummy in dummies:...
        score, winner = self.rashembo_match2(pop, agent=None)
        wins += winner
        robots.append((winner, "greenberg", score))
        score, winner = self.rashembo_match3(pop, agent=None)
        wins += winner
        robots.append((winner, "iocaine", score))
        population[index].fitness = wins / (len(dummies) + 2)
        population[index].robots_score = robots
```

The function fitness uses above function on all agents in the population affectively calculating fitness for all of them :

```
def fitness(self):
    self.fit_dummies(self.population, self.dummies)
```

Mate :

1. selects elite society
2. uses cross that we changed as 1.mutalism 2. communalism 3.parasitism

```
def mate(self, gen, fitnesstype, mut_type, prob_spec, population):
    esize = self.serviving_genes(gen, population)
    self.cross(esize, gen, population, len(population) - esize, fitnesstype, mut_type, prob_spec)

def cross(self, esize, gen, population, birth_count, fitnesstype, mut_type, prob_spec):
    self.mutualism()
    self.communalism(esize, birth_count)
    self.parasitism(esize, birth_count, gen)
```

Note: this algorithm changes the population dynamically and doesn't put them in a buffer

By now we explained the full algorithm of coevolution

Now we need to show the actual match between all participants:

We created a roshambo match class :

1. start match creates a match between an agent vs dummies and experts
2. match_X_Y: creates a match between members of X and members of Y
3. all_out_war : creates a match between all participants

```
def all_out_war(self,individual):
    agent=self.Start_match(individual)
    reslts=agent.robots_score
    all_results=[[] for _ in range(len(self.dummies)+len(self.experts)+1)]
    all_scores=[[] for _ in range(len(self.dummies)+len(self.experts)+1)]
    iocane, dummies1 = self.match_dummies_iocain()
    dvd=self.dummies_vs_dummies()
    greenberg, dummies2 = self.match_dummies_greenberg()

    #dummies:
    for i in range(12):
        all_results[i]=[score[0] for score in dvd[i]]+[dummies1[i][0]]+[dummies2[i][0]]+[reslts[i][0]]
        all_scores[i]=[score[2] for score in dvd[i]]+[dummies1[i][2] ]+[dummies2[i][2]]+[reslts[i][2]]
    all_results[12]=[green[0] for green in greenberg ]+[reslts[12][0]]
    all_results[13]=[ioc[0] for ioc in iocane]+[reslts[13][0]]
    all_results[14]=[res[0] for res in reslts]

    all_scores[12]=[green[2] for green in greenberg ]+[reslts[12][0]]
    all_scores[13]=[ioc[2] for ioc in iocane]+[reslts[13][0]]
    all_scores[14]=[res[2] for res in reslts]

    final_results=[sum(res)/len(res) for res in all_results]
    final_score=[sum(score)/len(score) for score in all_scores]
    return final_results,final_score
```

Sections to answer :

Section 1,2,3,4 :

Agent	Section1: Deterministic/Stochastic	Section 2/4: 1. Learn/predict 2. Meta Strategic/Random/Set of moves /Mix	Section 3: Exploration/Exploitation
Anti Flat Player	Deterministic	1. Prediction 2.	Exploitation
Copy Player	Deterministic : always plays according to the last move made by the opponent	1. learns : opponents last move 2.Uses Meta strategy	Exploitation
Freq Player	Deterministic : checks the most frequent move of the opponent and plays the winning move against it	1. Prediction: tries to predict next move based on former moves 2.learns: the most frequent move	Mix of both
Flat Player	Stochastic	1.doesn't learn or predict 2.Random : uses a coin flip to determine next move	Mix of both
Foxtrot Player	Stochastic	1.doesn't learn or predict 2.Random : uses a coin flip to determine next move	fairly Explorative
Bruijn 81 Player	Deterministic :uses constant set of moves	1. doesn't learn or predict 2.Constant set of moves	Neither
Pi Player	Deterministic : Pi is constant thus the actions are deterministic	1. doesn't learn or predict 2.Can be reduced to a constant set of moves	Partially exploitive
226 Player	Stochastic : is based on 20% 20% 60% distribution	1. doesn't learn or predict 2.Mix of strategies : uses Randomness with preset probabilities	Mix of both
Random Player	Stochastic: uses randomness to make moves	1. doesn't learn or predict 2.Random: uses randomness to make moves	Partially explorative
Rotating Player	Deterministic : it follow a pattern and knows what move to make based on that pattern	1. predict's the next move from the simple strategy that it follows 2.uses a strategy but not a meta one	Partially explorative

Switching Player	Stochastic	1. Learns from past moves made by the opponent 2. Uses Randomness	Exploitation
Switch a Lot Player	Stochastic : works like Switching Player but sometimes uses it's previous course of action	1. Learns from past moves : as it might use a past move that it made 2. Uses Randomness	Exploitation
Greenberg	Stochastic : uses Randomness	1. Learns from past moves and predicts next move 2. Uses Randomness	Both as it learns and predicts
Iocaine	Deterministic	1. Learns from past moves 2. Uses meta-strategy	Both ,same reason as above

Section 5 :

As we stated before , our population is comprised of Genes , each Gene contains an array of size 1000 of randomly created moves , we added a new Class named RPS that uses agent as a father Class , we changed the character creation function so that each element of the array is comprised of either 0,1,2 respectively :

```python
class RPS(DNA):
    def __init__(self):
        DNA.__init__(self)
        self.robots_score=[]
    def character_creation(self, target_size):
        return random.randint(0, 2)
```

The resulting agent looks something like this (after it gets through all the stages in the CO-Evolution-GA):

```
------Mutualizm faze-----------
------Communalism faze-----------
------Parasitism faze-----------
 Best:2,0,1,2,0,1,1,0,2,0,2,2,1,2,1,1,2,0,2,0,2,1,0,1,0,2,1,0,0,1,0,2,1,1,1,1,2,2,2,1,0,1,1,1,0,1,0,2,1,0,2,2,1,2,2,1,2,1,1,0,0,2,1,1,1
fittness: 0.8571428571428571  Time :  25494.9347682  ticks: 25494.92188668251
```

Section 6+7:

We already explained in the first couple of pages how the flow of the algorithm is and how the fitness function works

Note: The stoppage criteria of the algorithm is that fitness =1 or max iterations reached

Section 8:

How we couped with the resulting problems associated with co-evolution algorithm:

1.we used adaptive decrease to allow for gradual exploration-exploitation ratios which helped with the three given problems

2. we made sure that in each stage of the algorithm to change the most fit Gene only when we find a better suiter to the position witch helped with the circularity problem .

Section 9:

Our agent :

Agent	Section1: Deterministic/Stochastic	Section 2/4: 1. Learn/predict 2. Meta Strategic/Random/Set of moves /Mix	Section 3: Exploration/Exploitation
Our agent	Deterministic : it's true that we used randomness to create our agent but after the evolution process it sums up to be a constant set	1. doesn't predict nor use a meta-strategy 2. Static set of moves	None

Section 10:

In the folder output you can find all the relevant results of all simulations done by us ,one of them is the following results :

We recorded the results of 20 matches (in the code you can choose the number)

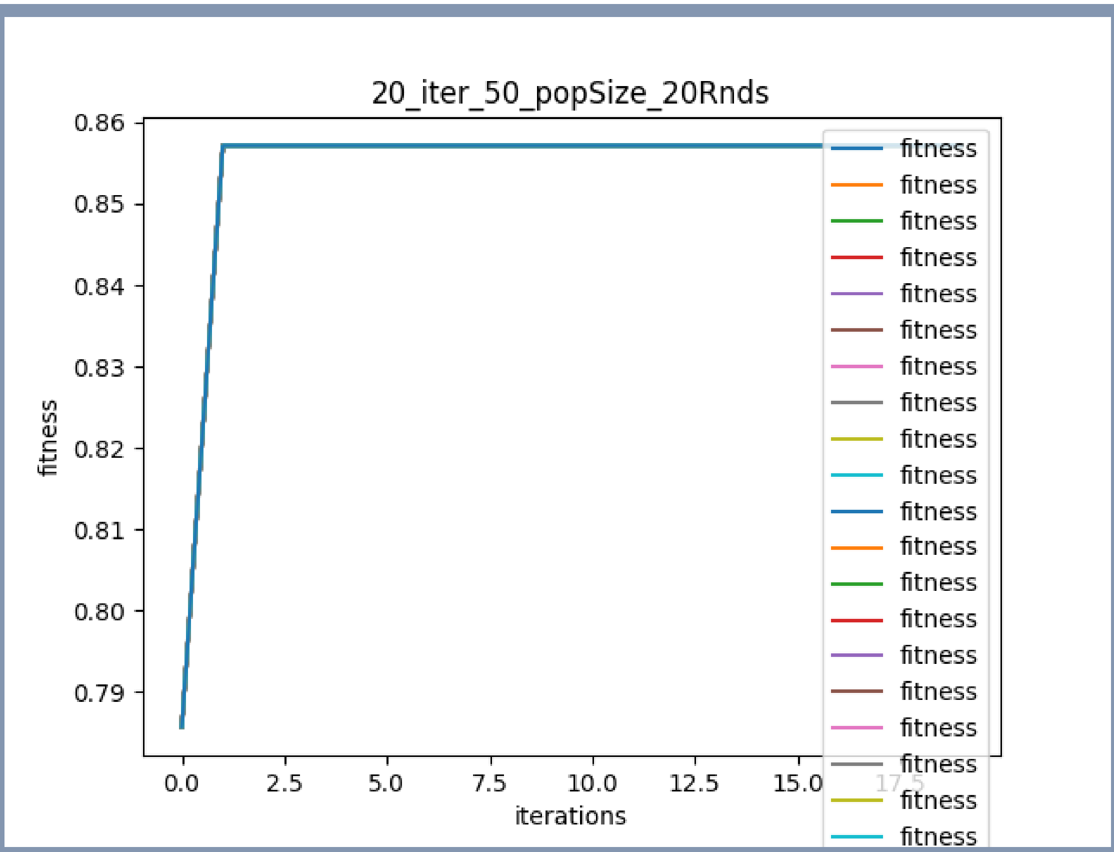
The numbers in the last row are the averages of all above values
We tried using evolution with fitness calculated as :

1. Tournament with all Agents (took too much time but we did it just for testing)
2. Tournament only with Dummies

Results of Tournament with all Agents as fitness function:

Our Agent's win Ratio	Anti Flat Player		Copy Player		Freq Player		Flat Player		Foxtrot Player		Bruijn 81 Player		Pi Player	
	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score
71%	True	43	True	19	True	6	False	-13	True	25	True	13	True	36
78%	True	42	True	19	True	6	False	-17	False	-6	True	13	True	36
57%	True	44	True	19	True	6	False	-19	False	-2	True	13	True	36
64%	True	44	True	19	True	6	False	-27	True	6	True	13	True	36
78%	True	43	True	19	True	6	True	16	True	49	True	13	True	36
64%	True	46	True	19	True	6	True	29	True	10	True	13	True	36
57%	True	39	True	19	True	6	True	19	False	-19	True	13	True	36
64%	True	48	True	19	True	6	True	18	False	-46	True	13	True	36
57%	True	39	True	19	True	6	False	-26	True	28	True	13	True	36
71%	True	38	True	19	True	6	True	36	False	-19	True	13	True	36
71%	True	44	True	19	True	6	False	-4	False	-17	True	13	True	36
71%	True	40	True	19	True	6	True	14	True	13	True	13	True	36
64%	True	41	True	19	True	6	True	12	False	-10	True	13	True	36
57%	True	44	True	19	True	6	True	32	True	39	True	13	True	36
64%	True	41	True	19	True	6	True	7	False	-3	True	13	True	36
64%	True	44	True	19	True	6	True	28	False	-1	True	13	True	36
64%	True	42	True	19	True	6	False	-19	False	-36	True	13	True	36
78%	True	42	True	19	True	6	False	-4	True	18	True	13	True	36
71%	True	40	True	19	True	6	False	-2	True	1	True	13	True	36
85%	True	42	True	19	True	6	True	11	True	10	True	13	True	36
Our Agent's win Ratio	Anti Flat Player		Copy Player		Freq Player		Flat Player		Foxtrot Player		Bruijn 81 Player		Pi Player	
67%	100		100		100		55		50		100		100	

226 Player		Random Player		Rotating Player		Switching Player		Switch a Lot Player		greenberg		iocaine	
winer	score	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score
False	-35	True	31	True	24	True	24	True	31	False	-6	False	-3
True	1	True	26	True	24	True	0	True	31	False	-8	True	3
False	-17	True	37	True	24	False	-32	True	12	False	-66	False	-12
False	-30	False	-5	True	24	False	-7	True	42	False	-21	True	9
False	-17	False	-29	True	24	True	25	True	8	False	-16	True	9
False	-40	False	-8	True	24	False	-4	True	20	False	-2	False	-2
False	-42	False	-25	True	24	True	6	False	-25	False	-28	False	-17
False	-25	True	30	True	24	False	-3	False	-22	True	5	False	-44
False	-19	False	-12	True	24	True	2	False	-13	False	-11	False	-51
False	-28	False	-27	True	24	False	-8	True	19	True	21	True	7
True	4	False	-16	True	24	True	0	True	54	True	41	False	-9
False	-34	False	-21	True	24	True	13	False	-18	True	3	False	-25
False	-62	False	-42	True	24	False	-11	True	4	False	-25	True	11
False	-43	False	-22	True	24	False	-6	False	-29	False	-35	False	-52
False	-6	False	-5	True	24	True	22	False	-25	False	-38	True	10
False	-31	True	28	True	24	False	-16	True	7	False	-19	False	-34
False	-31	False	-37	True	24	True	14	False	-30	True	22	True	2
True	24	True	5	True	24	True	2	False	-12	True	17	False	-5
False	-75	False	-18	True	24	True	36	True	26	False	-19	True	1
False	-14	True	31	True	24	True	38	False	-17	True	17	True	15
226 Player		Random Player		Rotating Player		Switching Player		Switch a Lot Player		greenberg		iocaine	
15		35		100		60		55		35		45	



Summary of above values :

Opponent	Agent Wins on average:
Anti Flat Player	100%
Copy Player	100%
Freq Player	100%
Flat Player	55%
Foxtrot Player	50%
Bruijn 81 Player	100%
Pi Player	100%
226 Player	15%
Random Player	35%
Rotating Player	100%
Switching Player	60%
Switch a Lot Player	55%
Greenberg	35%
Iocaine	45%

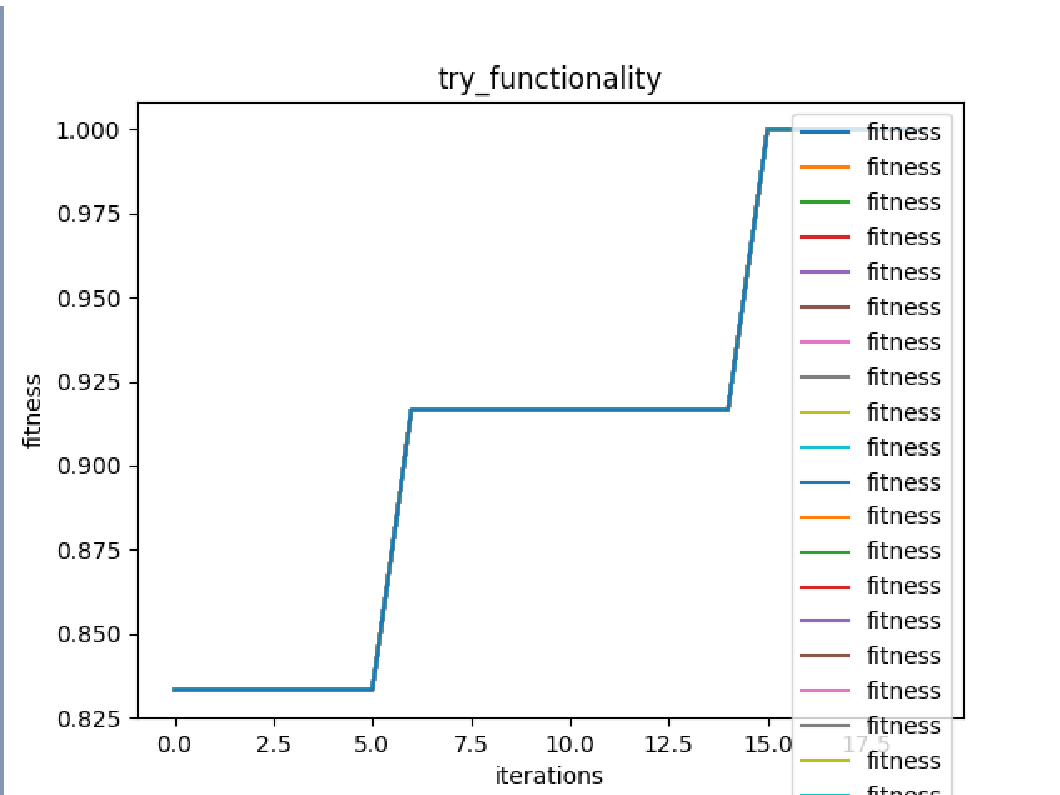
And its average win ratio is 67% i.e wins against 67% of it's opponents

Results of Tournament only with Dummies (2) :

|Results:|

Our Agent's win Ratio	Anti Flat Player		Copy Player		Freq Player		Flat Player		Foxtrot Player		Bruijn 81 Player		Pi Player	
	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score
78%	True	0	True	4	True	37	True	58	False	-76	True	44	True	16
71%	True	2	True	4	True	37	True	30	False	-35	True	44	True	16
78%	True	0	True	4	True	37	False	-59	True	53	True	44	True	16
71%	True	4	True	4	True	37	True	70	True	28	True	44	True	16
71%	True	3	True	4	True	37	True	13	True	13	True	44	True	16
64%	True	3	True	4	True	37	False	-9	True	22	True	44	True	16
85%	True	2	True	4	True	37	False	-21	False	-34	True	44	True	16
71%	True	2	True	4	True	37	True	6	False	-12	True	44	True	16
57%	True	3	True	4	True	37	False	-12	False	-68	True	44	True	16
50%	True	4	True	4	True	37	False	-8	False	-40	True	44	True	16
85%	True	6	True	4	True	37	True	12	True	14	True	44	True	16
78%	True	2	True	4	True	37	True	31	True	37	True	44	True	16
85%	True	3	True	4	True	37	True	3	True	35	True	44	True	16
78%	True	0	True	4	True	37	True	12	False	-34	True	44	True	16
71%	True	4	True	4	True	37	True	26	False	-5	True	44	True	16
71%	True	3	True	4	True	37	False	-6	True	18	True	44	True	16
64%	True	0	True	4	True	37	True	38	True	12	True	44	True	16
64%	True	6	True	4	True	37	False	-13	False	-51	True	44	True	16
78%	True	4	True	4	True	37	False	-12	True	15	True	44	True	16
71%	True	3	True	4	True	37	False	-4	True	22	True	44	True	16
Our Agent's win Ratio	Anti Flat Player		Copy Player		Freq Player		Flat Player		Foxtrot Player		Bruijn 81 Player		Pi Player	
72%	100		100		100		55		55		100		100	

226 Player		Random Player		Rotating Player		Switching Player		Switch a Lot Player		greenberg		iocaine	
winer	score	winer	score	winer	score	winer	score	winer	score	winer	score	winer	score
True	5	True	22	True	31	True	10	False	-19	True	25	False	-24
False	-20	False	-39	True	31	False	-26	True	13	True	17	True	44
False	-23	True	16	True	31	True	21	True	21	True	7	False	-11
False	-5	False	-7	True	31	True	18	True	74	False	-20	False	-40
False	-10	False	-12	True	31	True	26	True	1	False	-15	False	-11
True	11	False	-26	True	31	True	23	False	-13	False	-8	False	-29
True	18	True	38	True	31	True	10	True	16	True	28	True	2
False	-39	False	-47	True	31	True	30	False	-58	True	40	True	3
False	-37	False	-15	True	31	False	-38	True	30	False	-32	True	9
False	-2	False	-18	True	31	False	-22	False	-21	False	-32	True	3
True	32	False	-11	True	31	True	26	True	31	True	31	False	-11
True	37	True	8	True	31	True	47	False	-17	False	-15	False	-6
True	32	False	-9	True	31	True	27	True	9	False	-42	True	42
True	0	True	12	True	31	True	20	False	-29	True	34	False	-3
True	1	True	3	True	31	True	13	False	-21	False	-14	False	-12
False	-36	True	12	True	31	True	11	False	-2	True	58	False	-12
False	-20	True	31	True	31	False	-17	False	-43	False	-9	False	-6
False	-51	True	39	True	31	False	-27	True	5	False	-3	True	31
True	0	True	25	True	31	True	19	False	-28	False	-56	True	10
False	-10	True	63	True	31	False	-20	True	8	False	-9	True	22
226 Player		Random Player		Rotating Player		Switching Player		Switch a Lot Player		greenberg		iocaine	
45		55		100		70		50		40		45	



Summary of above values:

Opponent	Agent Wins on average:	Previous margins
Anti Flat Player	100%	100%
Copy Player	100%	100%
Freq Player	100%	100%
Flat Player	55%	55%
Foxtrot Player	55%	50%
Bruijn 81 Player	100%	100%
Pi Player	100%	100%
226 Player	45%	15%
Random Player	55%	35%
Rotating Player	100%	100%
Switching Player	70%	60%
Switch a Lot Player	50%	55%
Greenberg	40%	35%
Iocaine	45%	45%

Wins on average 72% of the matches it plays and we have a 5% improvement over the evolution with all agents
moreover it improved against several opponent's one huge jump would be against 226 player and random player by 20 to 30 percent improvement on those to players

Note: we do think that if Greenberg and iocaine were faster we would be able to get better margins using the first approach as we would then increase population size and then get more diversity in the solution

Section 11:

First approach

The last tournament (All out war as we called it) was done 5 times so that we understand the behavior of our agent , we will post the last round here , the specific result is in "outputs\20_iter_50_popSize_20Rnds\res out"

As well as the previous results all are found in "outputs\20_iter_50_popSize_20Rnds"

```
 Best:2,0,1,2,0,1,1,0,2,0,2,2,1,2,1,1,2,0,2,0,2,1,0,1,0,2,1,0,0,1,0,2,1,1,1,1,2,2,2,1,0,1,1,1,0,1,0,2,1,0,2,2
1,2,1,1,0,1,0,1,0,2,2,0,1,0,2,1,2,2,1,1,2,1,1,0,2,2,2,2,0,1,1,2,2,2,1,1,1,1,2,2,1,0,1,2,1,1,2,2,1,1,1,2,2,0,0
fittness: 0.8571428571428571  Time :  25494.9347682  ticks: 25494.92188668251
------Mutualizm faze-----------
------Communalism faze-----------
------Parasitism faze-----------
 Best:2,0,1,2,0,1,1,0,2,0,2,2,1,2,1,1,2,0,2,0,2,1,0,1,0,2,1,0,0,1,0,2,1,1,1,1,2,2,2,1,0,1,1,1,0,1,0,2,1,0,2,2
1,2,1,1,0,1,0,1,0,2,2,0,1,0,2,1,2,2,1,1,2,1,1,0,2,2,2,2,0,1,1,2,2,2,1,1,1,1,2,2,1,0,1,2,1,1,2,2,1,1,1,2,2,0,0
fittness: 0.8571428571428571  Time :  26803.794323  ticks: 26803.781769752502
 number of generations :  19
 Best:2,0,1,2,0,1,1,0,2,0,2,2,1,2,1,1,2,0,2,0,2,1,0,1,0,2,1,0,0,1,0,2,1,1,1,1,2,2,2,1,0,1,1,1,0,1,0,2,1,0,2,2
1,2,1,1,0,1,0,1,0,2,2,0,1,0,2,1,2,2,1,1,2,1,1,0,2,2,2,2,0,1,1,2,2,2,1,1,1,1,2,2,1,0,1,2,1,1,2,2,1,1,1,2,2,0,0
fittness: 0.8571428571428571  Time :  26803.7948999  ticks: 26803.781769752502

[[(True, 'Anti Flat Player', 43), (True, 'Copy Player', 19), (True, 'Freq Player', 6), (False, 'Flat Player',
226 Player', -17), (True, 'Random Player', 37), (True, 'Rotating Player', 24), (False, 'Switching Player', -3
Anti Flat Player', 46), (True, 'Copy Player', 19), (True, 'Freq Player', 6), (True, 'Flat Player', 29), (True
26 Player', -25), (True, 'Random Player', 30), (True, 'Rotating Player', 24), (False, 'Switching Player', -3)
e, 'Anti Flat Player', 44), (True, 'Copy Player', 19), (True, 'Freq Player', 6), (False, 'Flat Player', -4),
Player', -62), (False, 'Random Player', -42), (True, 'Rotating Player', 24), (False, 'Switching Player', -11)
'Anti Flat Player', 44), (True, 'Copy Player', 19), (True, 'Freq Player', 6), (True, 'Flat Player', 28), (Fal
6 Player', 24), (True, 'Random Player', 5), (True, 'Rotating Player', 24), (True, 'Switching Player', 2), (Fa
```

```
Anti Flat Player's win ratio is 26% and it's average score is -71.26666666666667
Copy Player's win ratio is 33% and it's average score is -136.26666666666668
Freq Player's win ratio is 33% and it's average score is -198.46666666666667
Flat Player's win ratio is 26% and it's average score is -205.73333333333332
Foxtrot Player's win ratio is 26% and it's average score is -229.0
Bruijn 81 Player's win ratio is 33% and it's average score is -232.93333333333334
Pi Player's win ratio is 26% and it's average score is -233.33333333333334
226 Player's win ratio is 33% and it's average score is -260.73333333333335
Random Player's win ratio is 26% and it's average score is -260.06666666666666
Rotating Player's win ratio is 26% and it's average score is -326.26666666666665
Switching Player's win ratio is 26% and it's average score is -345.8666666666667
Switch a Lot Player's win ratio is 26% and it's average score is -347.8666666666667
greenberg's win ratio is 46% and it's average score is 0.38461538461538464
iocaine's win ratio is 100% and it's average score is 3218.153846153846
Our Agent's win ratio is 78% and it's average score is 10.0
```

Results:

if we arrange the win ratio of all agents :
1. Iocain

2.Our Agent

3. Greenberg

4.copy ,freq,266

5. all of the rest

Thus showing that our agent comes at second place after iocain which is an impressive result .

If we look at the full folder that contains the results we can see that in 5 grand tournaments our agent comes mostly second behind Iocain and in the rest of the cases comes 3rd behind iocain and Greenberg

For a deterministic Agent it performed amazingly

Second strategy :

Under file output with the name try_functionality folder :

```
 Best:1,2,0,1,2,1,2,0,1,0,0,2,0,1,2,0,0,1,2,2,2,0,0,0,1,1,0,0,0,0,2,2,2,1,1,2,0,0,1,0,0,2,1,2,1,
1,0,1,1,2,0,2,0,1,0,2,2,2,2,2,1,1,2,1,0,1,0,2,1,0,1,2,0,0,1,2,2,2,2,2,1,0,2,1,2,1,1,0,0,2,1,1,1,0,
fittness: 1.0  Time :  45.71490440000001  ticks: 45.71469497680664
------Mutualism phase-----------
------Communalism phase-----------
------Parasitism faze-----------
 Best:1,2,0,1,2,1,2,0,1,0,0,2,0,1,2,0,0,1,2,2,2,0,0,0,1,1,0,0,0,0,2,2,2,1,1,2,0,0,1,0,0,2,1,2,1,
1,0,1,1,2,0,2,0,1,0,2,2,2,2,2,1,1,2,1,0,1,0,2,1,0,1,2,0,0,1,2,2,2,2,2,1,0,2,1,2,1,1,0,0,2,1,1,1,0,
fittness: 1.0  Time :  48.2757244  ticks: 48.27584362030029
 number of generations :  19
 Best:1,2,0,1,2,1,2,0,1,0,0,2,0,1,2,0,0,1,2,2,2,0,0,0,1,1,0,0,0,0,2,2,2,1,1,2,0,0,1,0,0,2,1,2,1,
1,0,1,1,2,0,2,0,1,0,2,2,2,2,2,1,1,2,1,0,1,0,2,1,0,1,2,0,0,1,2,2,2,2,2,1,0,2,1,2,1,1,0,0,2,1,1,1,0,
fittness: 1.0  Time :  48.27623850000005  ticks: 48.27683997154236
```

One of the runs :

```
Anti Flat Player's win ratio is 53% and it's average score is -62.93333333333333
Copy Player's win ratio is 46% and it's average score is -133.06666666666666
Freq Player's win ratio is 53% and it's average score is -189.13333333333333
Flat Player's win ratio is 46% and it's average score is -198.06666666666666
Foxtrot Player's win ratio is 53% and it's average score is -220.4
Bruijn 81 Player's win ratio is 53% and it's average score is -223.33333333333334
Pi Player's win ratio is 53% and it's average score is -224.26666666666668
226 Player's win ratio is 46% and it's average score is -249.6
Random Player's win ratio is 53% and it's average score is -247.86666666666667
Rotating Player's win ratio is 46% and it's average score is -316.4
Switching Player's win ratio is 53% and it's average score is -328.8
Switch a Lot Player's win ratio is 46% and it's average score is -338.1333333333333
greenberg's win ratio is 30% and it's average score is -1.8461538461538463
iocaine's win ratio is 100% and it's average score is 3217.3846153846152
Our Agent's win ratio is 85% and it's average score is 22.357142857142858
```

Results:

if we arrange the win ratio of all agents :
1. Iocain

2.Our Agent

3. The rest of the agents

Thus showing that our agent comes at second place after iocain which is an impressive result .

If we look at the full folder that contains the results we can see that in 5 grand tournaments our agent comes mostly second behind Iocain and in only one case came 3rd behind iocain and Greenberg

For a deterministic Agent it performed amazingly

How to operate :

simply run the exe file and enter what is asked of you

Things to improve:

1. Create a meta-strategy to help improve our results , we wanted to use Markov chains , but couldn't due to time constraints ,and personal issues
2. Try to find a solution that isn't deterministic because one string of moves can't trump all possible agent's

Please note that there are more tests in the output file than what is shown here