

CM3020 Artificial Intelligence Midterm Coursework

Part B

Introduction

This report details the process and results of adapting a genetic algorithm for evolving creatures to complete the task of climbing a mountain. Building upon the existing Genetic Algorithm/Creatures case study, I integrated a new environment that features a mountain, challenging the creatures to climb as high as possible. The primary objective was to modify the genetic algorithm to evolve creatures capable of achieving this task. This involved adapting the fitness function to prioritize climbing ability, and conducting various experiments to optimize the creatures' performance. The following sections outline the methodology, experiments, and results achieved in this process.

Initial Creature Development

Before I created the creatures, I developed a new fitness function. This consisted of a few new functions in the Creature.py file such as, `get_height_reached()`, `get_stability()`, `get_distance_to_centre()`, `get_fitness()`. The `get_height_reached()` method calculates the maximum height the creature reaches during the simulation by extracting the highest z-coordinate from the stored positions. The `get_stability()` method assesses the creature's stability by calculating the variance in lateral movement from the start position, using the inverse of the variance as a stability measure. The `get_distance_to_center()` method computes the Euclidean distance from the creature's last recorded position to the mountain's centre, assumed to be at the origin. Furthermore, I updated the `update_position()` function to include a reward and penalty system, where the fitness value would be increased when the creature moves closer to the mountain's centre and is decreased when it moves away.

Finally, using all the above functions, the `get_fitness()` method combines the height reached, the inverse distance to the centre, and the accumulated reward or penalty to determine the overall fitness of the creature. These enhancements effectively guide the evolutionary process towards developing creatures that excel at climbing the mountain.

```
def get_height_reached(self):
    if self.positions:
        heights = [pos[2] for pos in self.positions]
        return max(heights)
    return 0

def get_stability(self):
    if len(self.positions) < 2:
        return 0
    deviations = [math.sqrt((pos[0] - self.start_position[0])**2 + (pos[1] - self.start_position[1])**2) for pos in self.positions]
    stability = 1 / (1 + np.var(deviations)) # Inverse of variance as a measure of stability
    return stability

def get_distance_to_center(self, pos):
    center = np.array([0, 0, 0]) # Assuming the mountain center is at the origin
    return np.linalg.norm(np.array(pos) - center)

def get_fitness(self):
    distance_travelled = self.get_distance_travelled()
    height_reached = self.get_height_reached()
    stability = self.get_stability()
    return distance_travelled + height_reached + stability + self.fitness # Include the reward/penalty component
    # return distance_travelled + height_reached + self.fitness # Include the reward/penalty component
```

```
def update_position(self, pos):
    if self.start_position is None:
        self.start_position = pos
    else:
        self.last_position = pos
    self.positions.append(pos)

    # Update fitness based on movement towards or away from the mountain center
    distance_to_center = self.get_distance_to_center(pos)
    if self.previous_distance_to_center is not None:
        if distance_to_center < self.previous_distance_to_center:
            self.fitness += 1/24 # Reward for moving closer
        else:
            self.fitness -= 1/24 # Penalty for moving away
    self.previous_distance_to_center = distance_to_center
```

To create the creatures, I experimented with many different settings and eventually created four different creatures, each with unique characteristics and stats. The initial phase involved experimenting with different configurations to understand how various attributes affect the creatures' ability to climb the mountain. Below, I have shown images of code modifications made to create these four distinct creatures. In the images shown, I have documented the progress of each creature as it underwent the evolution.

Creature 1

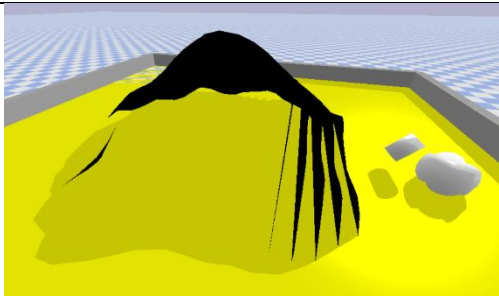
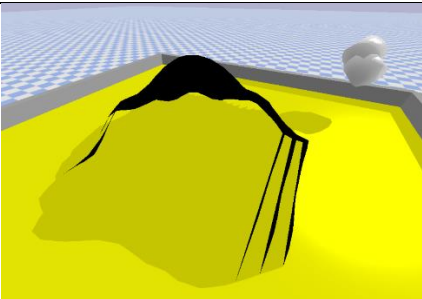
- Using populations size of 30 and gene count of 4 for 300 iterations

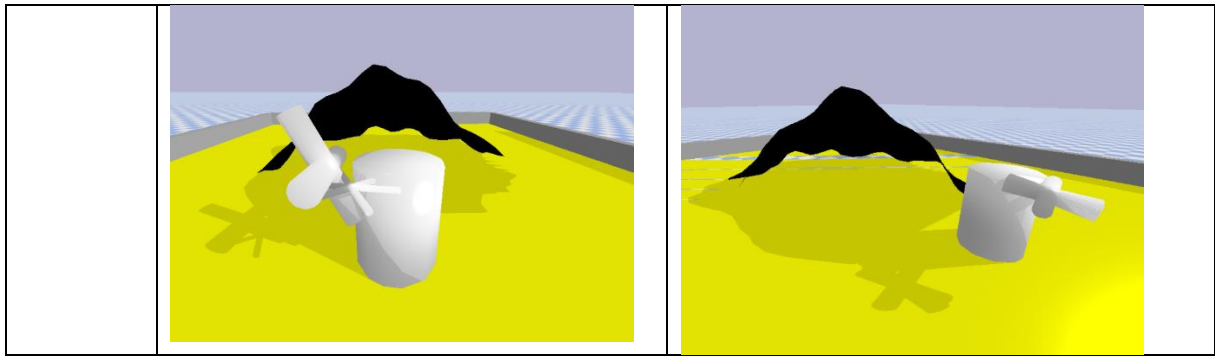
```
pop = population.Population(pop_size=30, gene_count=4)
sim = simulation.Simulation()

for iteration in range(300):
    for cr in pop.creatures:
        cr.reset_position_history() # Reset position history for stability calculation
        sim.run_creature(cr, 2400)
```

- Adjusting the Motor class to set a different amplitude and frequency

```
class Motor:
    def __init__(self, control_waveform, control_amp, control_freq):
        if control_waveform <= 0.5:
            self.motor_type = MotorType.PULSE
        else:
            self.motor_type = MotorType.SINE
        # self.amp = control_amp
        # self.freq = control_freq
        self.amp = 30
        self.freq = 30
        self.phase = 0
```

Iteration:	10	100
		
	200	300



Creature 2

- Using populations size of 25 and gene count of 8 for 300 iterations

```
pop = population.Population(pop_size=25, gene_count=8)
sim = simulation.Simulation()

for iteration in range(300):
    for cr in pop.creatures:
        cr.reset_position_history() # Reset position history for stability calculation
        sim.run_creature(cr, 2400)
```

- Adjusting the genome specs to try and achieve a rounder shape as my earlier shapes were too cylindrical

```
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":2},
        "link-radius": {"scale":2},
        "link-recurrence": {"scale":3},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1},
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1},
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":1},
        "joint-origin-xyz-2":{"scale":1},
        "joint-origin-xyz-3":{"scale":1},
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.25},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec
```

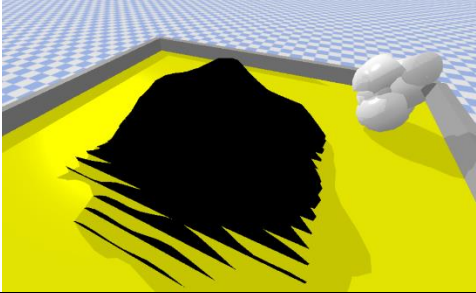
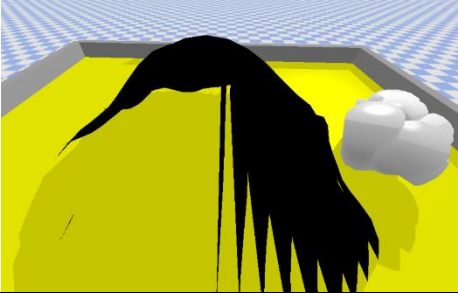
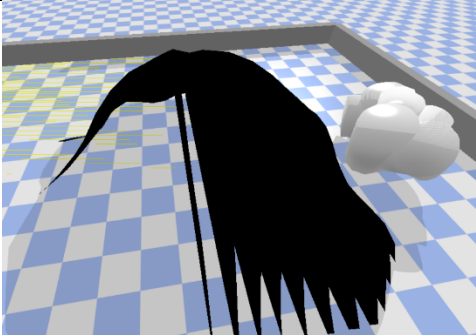
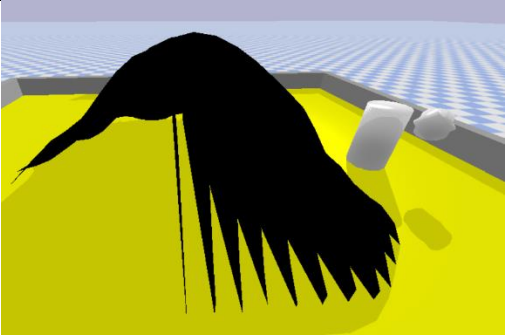
- Adjusted the rewards and penalty function to have a more severe impact on the fitness value

```

def update_position(self, pos):
    if self.start_position is None:
        self.start_position = pos
    else:
        self.last_position = pos
    self.positions.append(pos)

    # Update fitness based on movement towards or away from the mountain center
    distance_to_center = self.get_distance_to_center(pos)
    if self.previous_distance_to_center is not None:
        if distance_to_center < self.previous_distance_to_center:
            self.fitness += 1 # Reward for moving closer
        else:
            self.fitness -= 1 # Penalty for moving away
    self.previous_distance_to_center = distance_to_center

```

Iteration:	10	100
		
	200	300
		

Creature 3

- Using populations size of 25 and gene count of 5 for 300 iterations

```

pop = population.Population(pop_size=25, gene_count=5)
sim = simulation.Simulation()

for iteration in range(300):
    for cr in pop.creatures:
        cr.reset_position_history() # Reset position history for stability calculation
        sim.run_creature(cr, 2400)

```

- Adjusting the Motor class to try a different amplitude and frequency


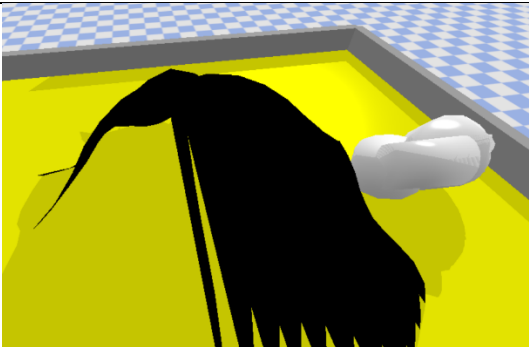
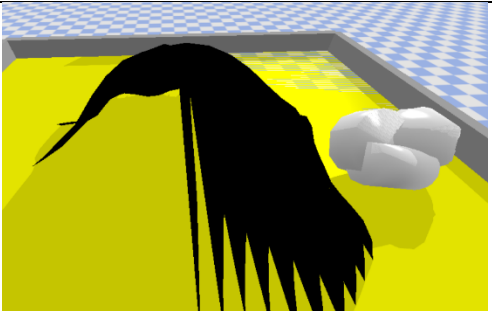
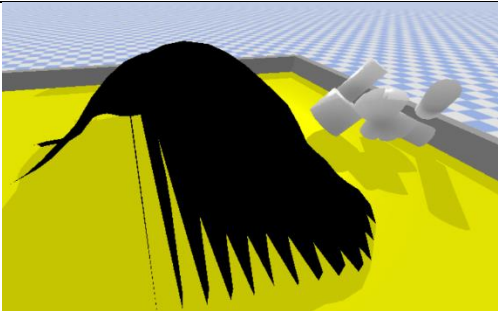
```
class Motor:
    def __init__(self, control_waveform, control_amp, control_freq):
        if control_waveform <= 0.5:
            self.motor_type = MotorType.PULSE
        else:
            self.motor_type = MotorType.SINE
        # self.amp = control_amp
        # self.freq = control_freq
        self.amp = 10
        self.freq = 10
        self.phase = 0
```

- Changing the fitness function to use the inverse distance to the centre of the mountain rather than the distance travelled by the object to calculate the fitness value

```
def get_fitness(self):
    distance_to_center = self.get_distance_to_center()
    height_reached = self.get_height_reached()
    stability = self.get_stability()

    # Inverse distance to center (higher value for closer distance)
    if distance_to_center == 0:
        inverse_distance_to_center = float('inf') # Avoid division by zero
    else:
        inverse_distance_to_center = 1 / distance_to_center

    return stability + height_reached + inverse_distance_to_center + self.fitness # Include the reward/penalty component
```

Iteration:	10	100
		
	200	300
		

Creature 4

- Using populations size of 25 and gene count of 5 for 300 iterations

```
pop = population.Population(pop_size=25, gene_count=5)
sim = simulation.Simulation()

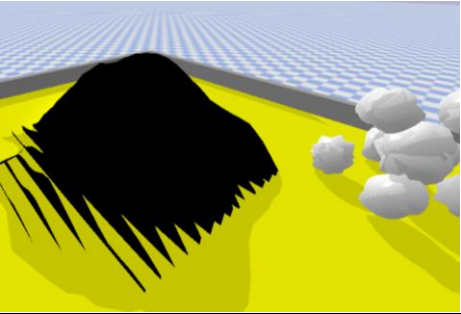
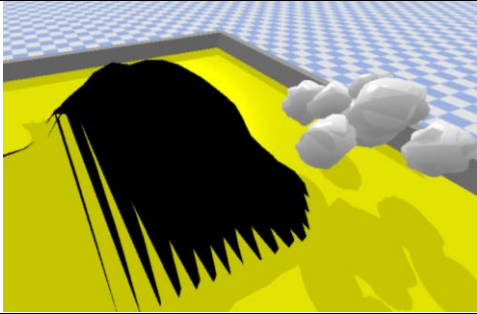
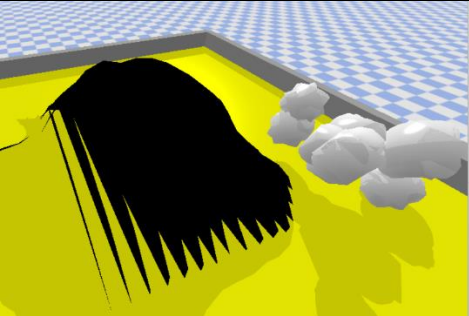
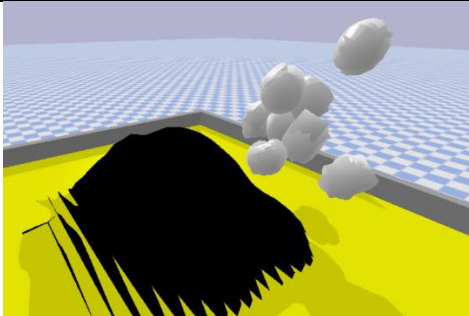
for iteration in range(300):
    for cr in pop.creatures:
        cr.reset_position_history() # Reset position history for stability calculation
        sim.run_creature(cr, 2400)
```

- Adjusting the point mutate rates to see its effect

```
# Create fitness map for parent selection
fit_map = population.Population.get_fitness_map(fits)
new_creatures = []
for i in range(len(pop.creatures)):
    p1_ind = population.Population.select_parent(fit_map)
    p2_ind = population.Population.select_parent(fit_map)
    p1 = pop.creatures[p1_ind]
    p2 = pop.creatures[p2_ind]
    # now we have the parents!
    dna = genome.Genome.crossover(p1.dna, p2.dna)
    dna = genome.Genome.point_mutate(dna, rate=0.2, amount=0.25)
    dna = genome.Genome.shrink_mutate(dna, rate=0.25)
    dna = genome.Genome.grow_mutate(dna, rate=0.1)
    cr = creature.Creature(1)
    cr.update_dna(dna)
    new_creatures.append(cr)
```

- Adjusting the genome specs to try and change the Z position of the joints as the joints were always forming to high above the ground level

```
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":1}, # Adjust length scale as needed
        "link-radius": {"scale":1},
        "link-recurrence": {"scale":3},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1}, # This will be interpreted as revolute in URDFLink
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1}, # Scale covers range for controlling axis (0.0, 0.5, 1.0)
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":1},
        "joint-origin-xyz-2":{"scale":1},
        "joint-origin-xyz-3":{"scale":2}, # Scale for Z position adjustment
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.25},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec
```

Iteration:	10	100
		
	200	300
		

Selection of the Optimal Creature

After evaluating the performance of all four creatures, I selected the one that demonstrated the best climbing ability to continue with further experiments. I chose Creature 4 as it had exhibited the most promising traits and adaptability to the mountain environment.

Experimental Setup and Results

Using the selected creature, I conducted several experiments to optimize the genetic algorithm's performance. These experiments involved changing population sizes, gene counts, fitness functions, and genome codes. Each experiment aimed to identify the configurations that maximize the creature's climbing ability.

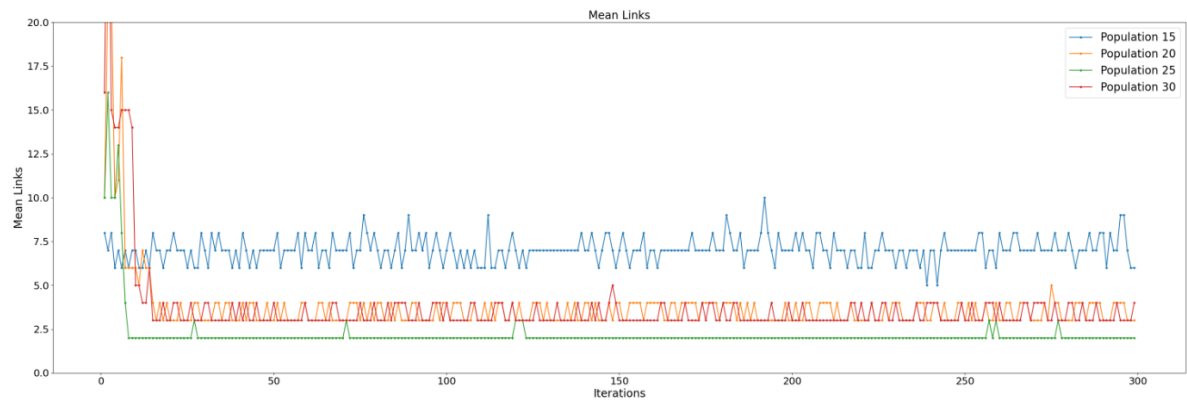
Experiment 1: Population Size

Population Configurations Tested: 15, 20, 25, 30

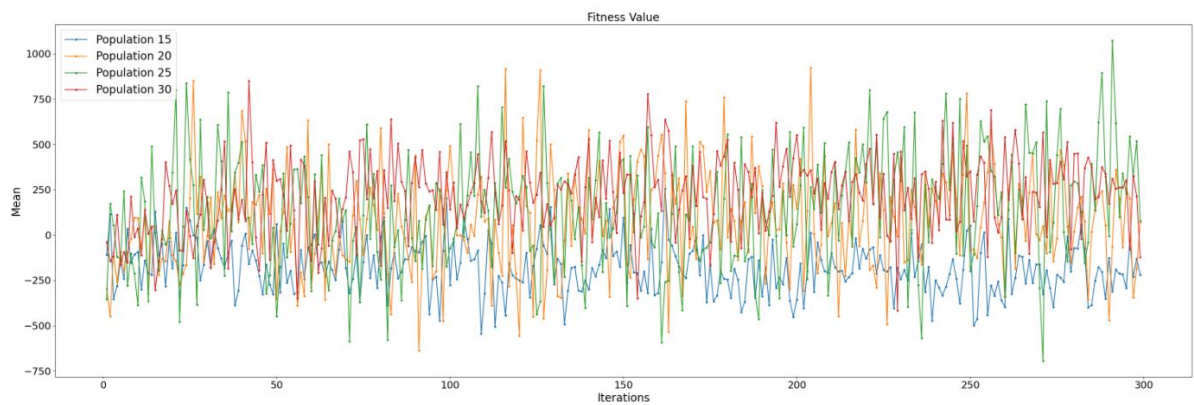
Iterations: 300

Results:

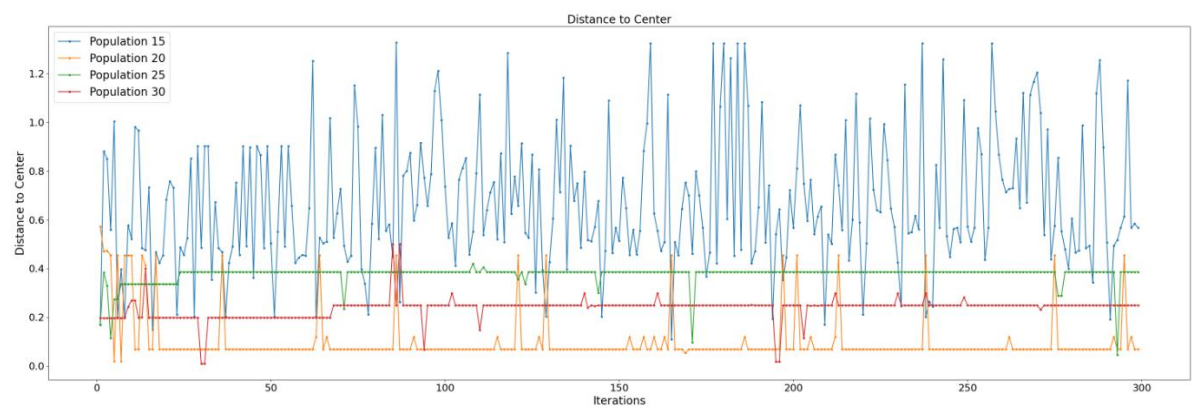
Mean Links:



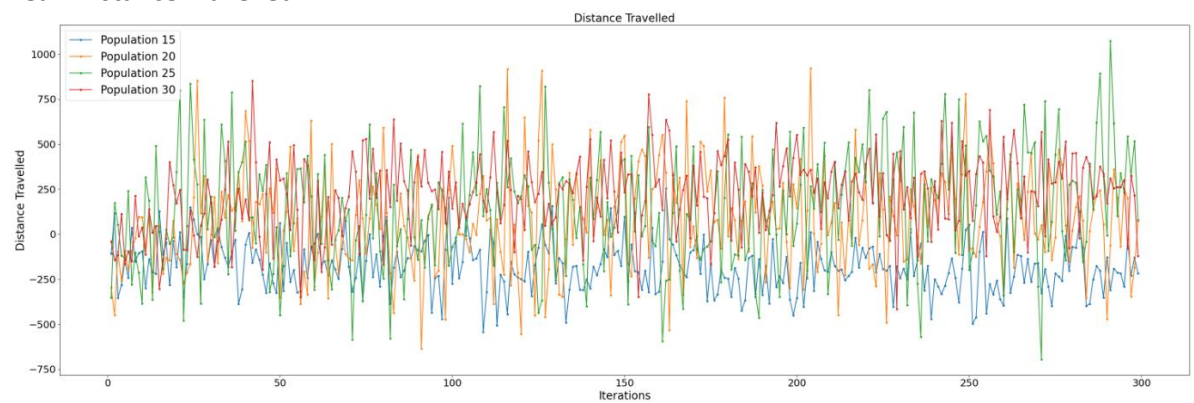
Fitness Value:



Minimum Distance from the Centre:



Mean Distance Travelled:



I then calculated the average result values per 100 iterations for each population size to ensure I had a clear understanding of the data since the graphs cannot offer such in depth insight.

Population size 15:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	50.5	-147.256410	6.990000	11.770000	0.628060	-147.256410
101-200	150.5	-192.048830	7.040000	12.480000	0.672720	-192.048830
201-300	250.0	-204.374374	7.040404	12.161616	0.691475	-204.374374

Population size 20:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	50.5	24.806560	4.340000	9.510000	0.12452	24.806560
101-200	150.5	123.964190	3.510000	7.750000	0.08774	123.964190
201-300	250.0	94.208293	3.474747	7.222222	0.09097	94.208293

Population size 25:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	50.5	69.52764	2.590000	5.580000	0.368300	69.52764
101-200	150.5	99.61598	2.030000	4.320000	0.381970	99.61598
201-300	250.0	255.40304	2.030303	4.525253	0.380576	255.40304

Population size 30:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	50.5	152.149180	4.590000	12.030000	0.218590	152.149180
101-200	150.5	249.003920	3.260000	8.820000	0.244730	249.003920
201-300	250.0	252.300505	3.323232	8.525253	0.249263	252.300505

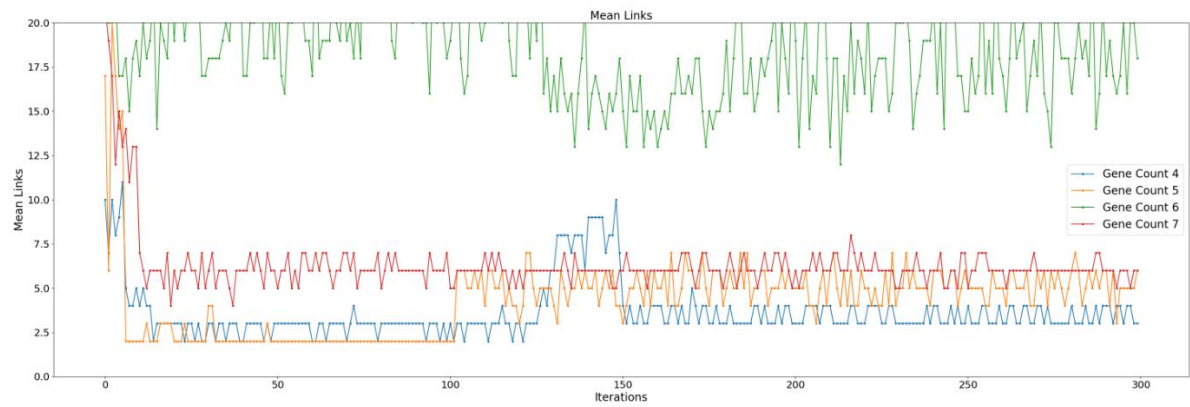
Viewing the data from both the graphs and tables, we can see that the mean links for the population size 15 is the highest. The links indicate the average complexity of the creatures' structures. A higher mean links value suggests that, on average, creatures have more parts and potentially more complex behaviors. However, its distance to centre is the highest. Whereas for the population size 20, it has the second highest mean links while having the lowest distance to the centre which is the most ideal as we want our creature to reach the mountain. Hence, I chose to use a population size of 20 going forward with further experiments.

Experiment 2: Gene Count

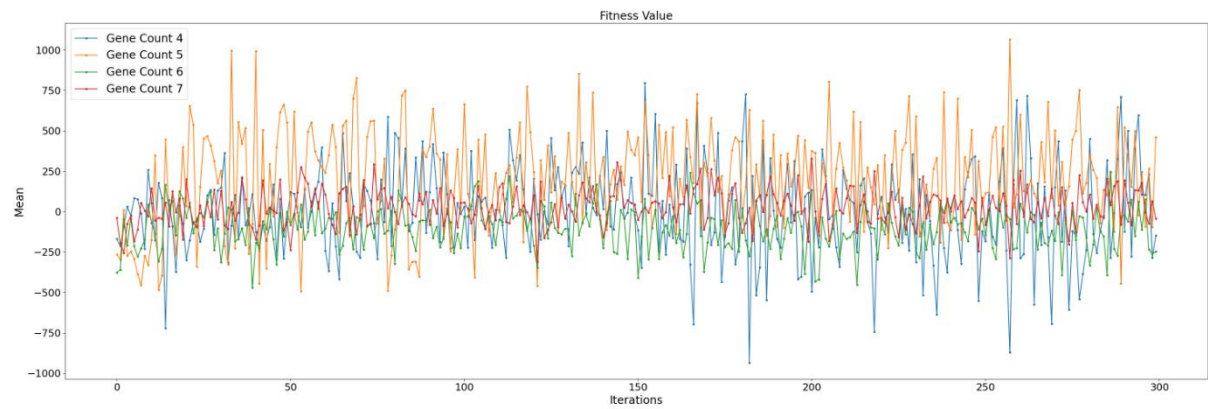
Gene Count Configurations Tested: 4,5,6,7
Iterations: 300

Results:

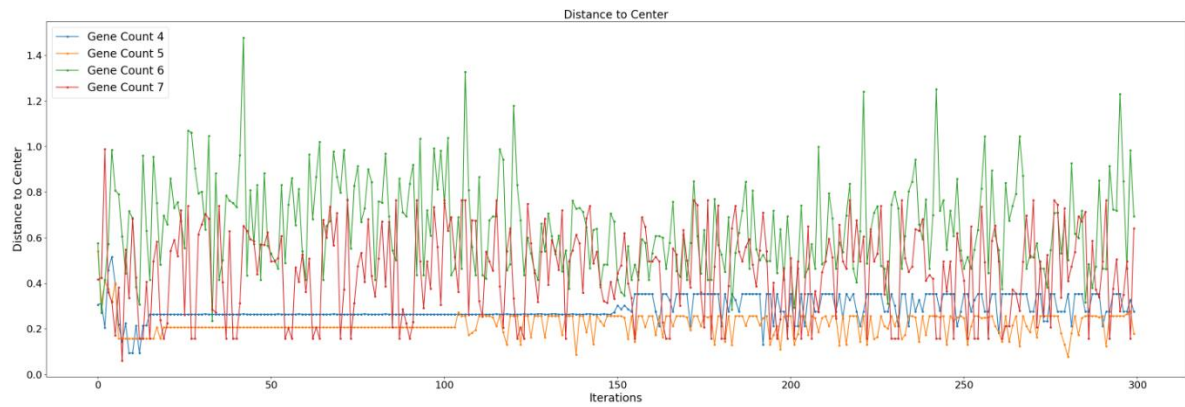
Mean Links:



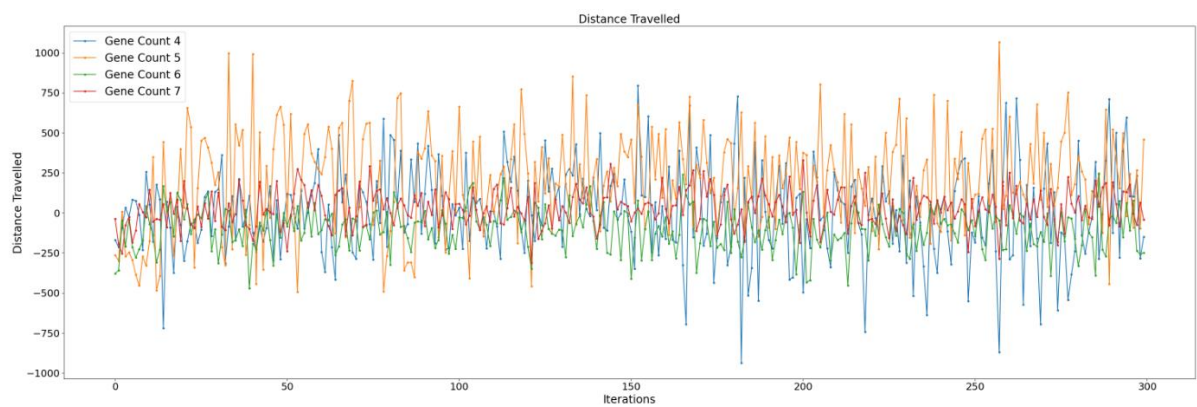
Fitness Value:



Minimum Distance from the Centre:



Mean Distance Travelled:



I then calculated the average result values per 100 iterations for each gene count configuration to ensure I had a clear understanding of the data since the graphs only show the general trend but not a very clear analysis.

Gene Count 4:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	0.85717	3.30	6.76	0.26158	0.85717
101-200	149.5	38.48668	4.26	9.78	0.29129	38.48668
201-300	249.5	-31.14567	3.41	7.52	0.31360	-31.14567

Gene Count 5:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	149.70542	2.88	6.58	0.21209	149.70542
101-200	149.5	227.61824	5.09	13.35	0.22746	227.61824
201-300	249.5	223.84058	5.11	12.58	0.22216	223.84058

Gene Count 6:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-100.94364	20.36	45.95	0.71636	-100.94364
101-200	149.5	-86.57118	17.59	37.64	0.59184	-86.57118
201-300	249.5	-112.79512	17.98	43.70	0.64411	-112.79512

Gene Count 7:

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	15.02846	6.91	12.57	0.43439	15.02846
101-200	149.5	33.66382	6.03	10.71	0.46920	33.66382
201-300	249.5	27.36366	6.08	11.09	0.44456	27.36366

From the data, we can observe that the gene count 6 creature had the highest mean number of links, potentially creating a more complex creature that can climb. However, its distance travelled is very high yet not having a close distance to the centre. The gene count 5 creature however has a steadily improving mean number of links, as well as a decreasing distance to the centre. Hence, I chose gene count 5 as the most suitable among the other options and used this gene count for future experimentations.

Experiment 3: Fitness Function

The fitness function I created is based on 4 values; Fitness value based on 4 things: The Height Reached, Stability, Distance from Centre of Mountain, Fitness Reward/Penalty. I have explained the purpose of each feature earlier in the report. In previous experiments, my fitness function has been using all 4 features. I chose to do this experiment to test out if the features have a significant effect on the creatures' evolutions

Configurations Tested:

1st: Fitness value calculated without height reached and stability functions

2nd: Fitness value calculated without stability

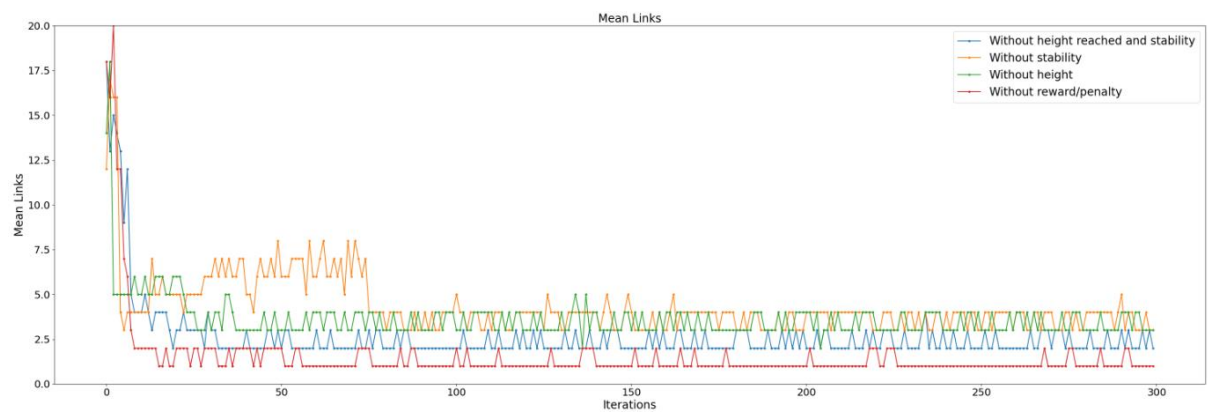
3rd: Fitness calculated without height

4th: Fitness calculated without reward/penalty component

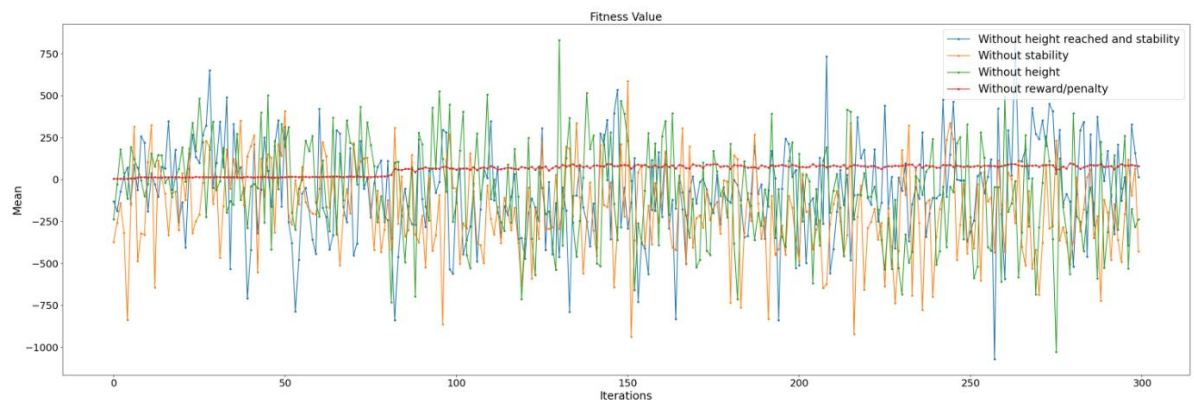
Iterations: 300

Results:

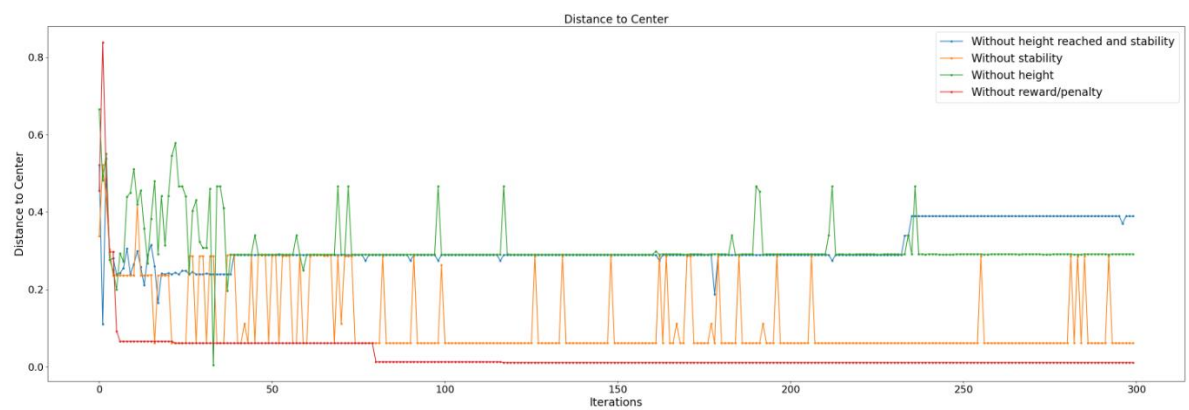
Mean Links:



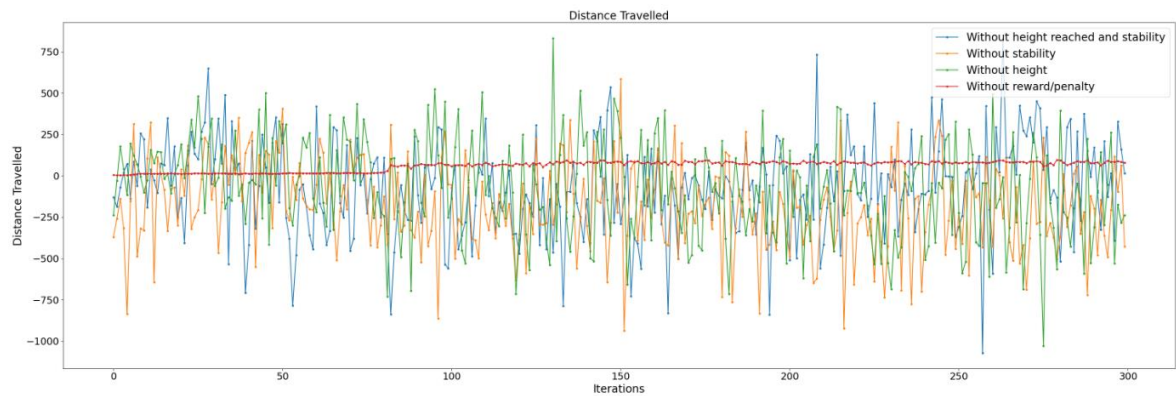
Fitness Value:



Minimum Distance from the Centre:



Mean Distance Travelled:



1st: Fitness value calculated without height reached and stability functions

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-58.30276	3.32	7.90	0.27705	-58.30276
101-200	149.5	-136.34922	2.30	5.82	0.28768	-136.34922
201-300	249.5	-12.13797	2.33	5.83	0.35466	-12.13797

2nd: Fitness value calculated without stability

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-105.07928	5.66	12.51	0.17974	-105.07928
101-200	149.5	-204.88783	3.68	7.59	0.08500	-204.88783
201-300	249.5	-246.46865	3.67	7.65	0.07450	-246.46865

3rd: Fitness calculated without height

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	27.5369	4.12	8.85	0.33313	27.5369
101-200	149.5	-95.3853	3.42	7.25	0.29600	-95.3853
201-300	249.5	-161.8676	3.37	7.78	0.29526	-161.8676

4th: Fitness calculated without reward/penalty component

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	23.03884	2.26	7.32	0.07318	23.03884
101-200	149.5	76.48178	1.13	3.29	0.01134	76.48178
201-300	249.5	80.10490	1.13	3.42	0.01100	80.10490

From the analysis, even though the minimum distance to the centre is the best when the rewards/penalty component is removed, it is inaccurate as the mean distance travelled is also close to zero as evidenced from the graph. Hence, choosing the next best option would be the

the creature made without including the stability function in the fitness value. I will use this new fitness function for future experiments.

Experiment 4: Genome Codes

For this experiment, I will test different genome encoding schemes.

Configurations Tested:

1st: Changing scaling factor of link length

```
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":3}, # Adjust length scale
        "link-radius": {"scale":1},
        "link-recurrence": {"scale":3},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1}, # This will be interpreted as revolute in URDFLink
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1}, # Scale covers range for controlling axis (0.0, 0.5, 1.0)
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":1},
        "joint-origin-xyz-2":{"scale":1},
        "joint-origin-xyz-3":{"scale":2}, # Scale for Z position adjustment
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.25},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec
```

2nd: Changing control amplitude

```
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":1}, # Adjust length scale
        "link-radius": {"scale":1},
        "link-recurrence": {"scale":3},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1}, # This will be interpreted as revolute in URDFLink
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1}, # Scale covers range for controlling axis (0.0, 0.5, 1.0)
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":1},
        "joint-origin-xyz-2":{"scale":1},
        "joint-origin-xyz-3":{"scale":2}, # Scale for Z position adjustment
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.75},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec
```

3rd: Changing link recurrence

```
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":1}, # Adjust length scale
        "link-radius": {"scale":1},
        "link-recurrence": {"scale":4},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1}, # This will be interpreted as revolute in URDFLink
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1}, # Scale covers range for controlling axis (0.0, 0.5, 1.0)
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":1},
        "joint-origin-xyz-2":{"scale":1},
        "joint-origin-xyz-3":{"scale":2}, # Scale for Z position adjustment
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.25},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec
```

4th: Changing joint origin XYZ values

```

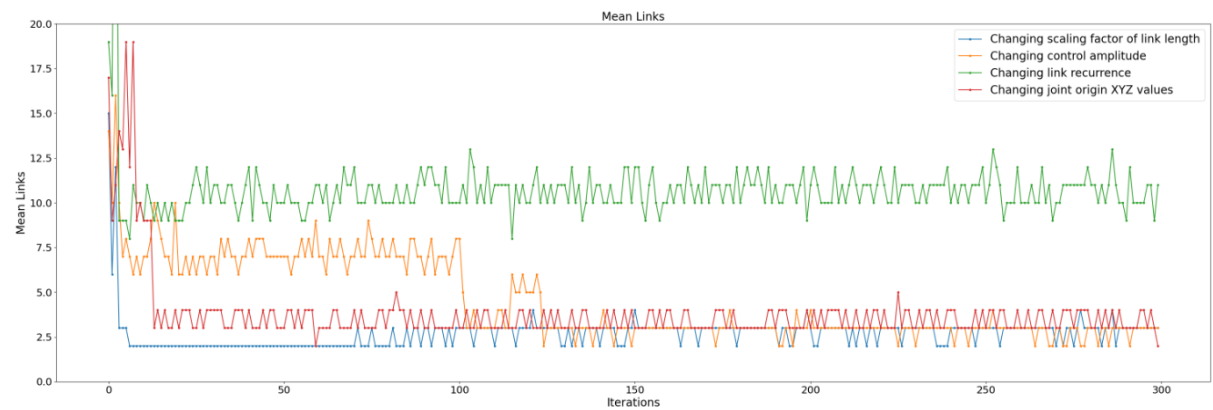
@staticmethod
def get_gene_spec():
    gene_spec = {"link-shape":{"scale":1},
        "link-length": {"scale":1}, # Adjust length scale
        "link-radius": {"scale":1},
        "link-recurrence": {"scale":3},
        "link-mass": {"scale":1},
        "joint-type": {"scale":1}, # This will be interpreted as revolute in URDFLink
        "joint-parent":{"scale":1},
        "joint-axis-xyz": {"scale":1}, # Scale covers range for controlling axis (0.0, 0.5, 1.0)
        "joint-origin-rpy-1":{"scale":np.pi * 2},
        "joint-origin-rpy-2":{"scale":np.pi * 2},
        "joint-origin-rpy-3":{"scale":np.pi * 2},
        "joint-origin-xyz-1":{"scale":2},
        "joint-origin-xyz-2":{"scale":2},
        "joint-origin-xyz-3":{"scale":3}, # Scale for Z position adjustment
        "control-waveform":{"scale":1},
        "control-amp":{"scale":0.25},
        "control-freq":{"scale":1}
    }
    ind = 0
    for key in gene_spec.keys():
        gene_spec[key]["ind"] = ind
        ind = ind + 1
    return gene_spec

```

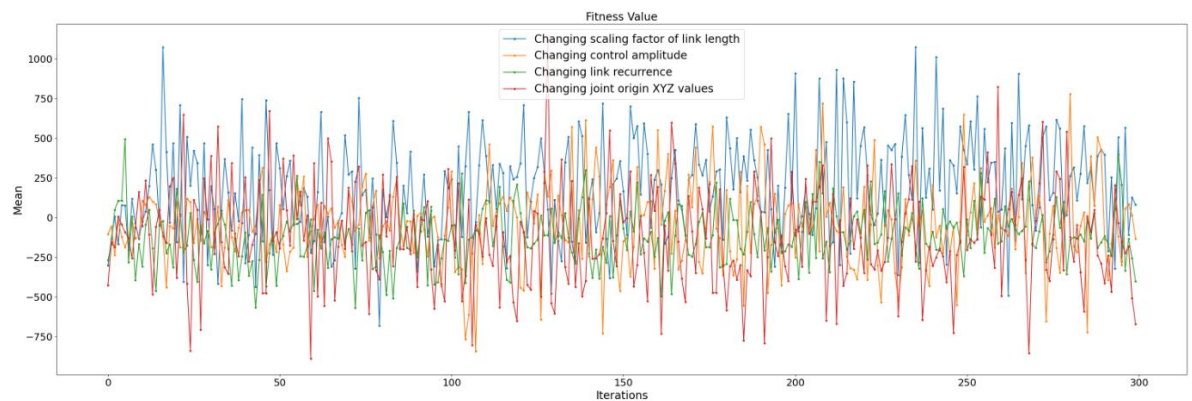
Iterations:300

Results:

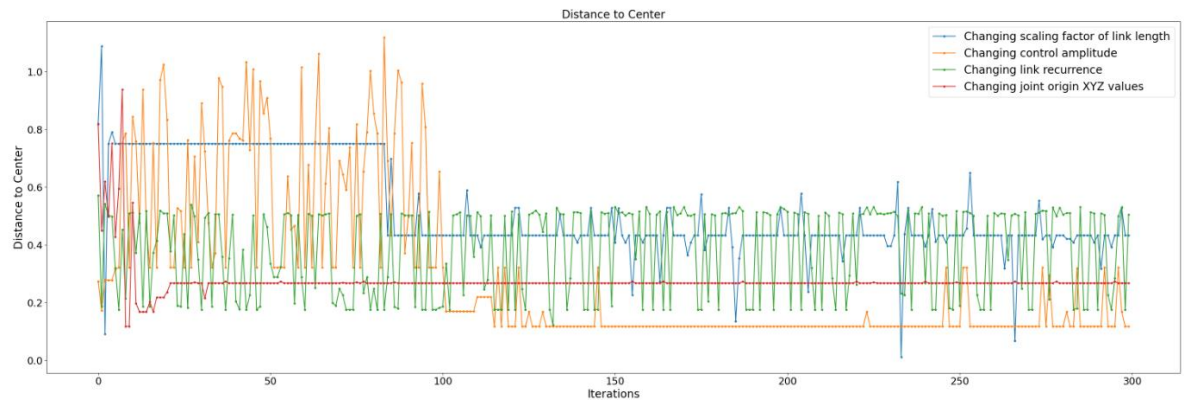
Mean Links:



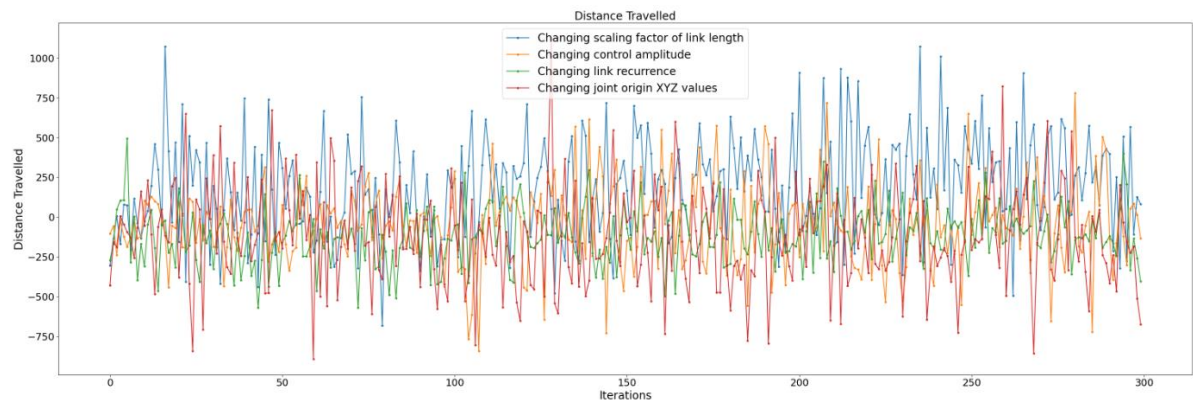
Fitness Value:



Minimum Distance from the Centre:



Mean Distance Travelled:



1st: Changing scaling factor of link length

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	95.73143	2.43	5.56	0.70211	95.73143
101-200	149.5	199.87317	2.86	6.32	0.43462	199.87317
201-300	249.5	268.23562	2.84	5.70	0.42749	268.23562

2nd: Changing control amplitude

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-48.69264	7.38	15.03	0.60095	-48.69264
101-200	149.5	-31.86706	3.25	6.80	0.13898	-31.86706
201-300	249.5	-40.18324	2.88	5.78	0.13553	-40.18324

3rd: Changing link recurrence

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-162.09660	10.74	20.45	0.35310	-162.09660
101-200	149.5	-110.45452	10.70	19.12	0.39586	-110.45452
201-300	249.5	-83.83758	10.71	19.97	0.39872	-83.83758

4th: Changing joint origin XYZ values

	fittest	mean	mean_links	max_links	dist_to_center	dist_travelled
1-100	49.5	-81.10906	4.61	10.69	0.28991	-81.10906
101-200	149.5	-157.25505	3.35	6.92	0.26815	-157.25505
201-300	249.5	-121.13229	3.47	7.83	0.26831	-121.13229

From the values and graphs, changing the control amplitude allowed the creature to have the smallest distance to the centre. However, it seems to stagnate whereas changing the scaling factor of the link length has a high distance travelled but is not close to the centre. Perhaps future experiments will require a combination of both configurations.

Conclusion

The experiments conducted provide valuable insights into optimizing genetic algorithms for specific tasks. The results demonstrate the effectiveness of various configurations in enhancing the creature's ability to climb the mountain. Further research and experimentation could explore additional parameters and advanced genetic encoding schemes to continue improving performance. This project showcases the potential of evolutionary algorithms in solving complex, dynamic tasks within simulated environments.