Deleted from literature review

##### Fitness Proportionate Selection (Roulette Selection)

Select individuals in proportion to their fitness. The higher the fitness, the more likely that we pick the individual. A ‘Size’ is given to an individual based on their fitness, so a higher fitness means a bigger ‘Size’. Then a number is picked between 0 and the total size of all the individuals. This number is essentially a pointer like on a roulette wheel, but the roulette wheel is made up of all the individuals and the fitter individuals have a bigger slice of the wheel, as such if the number lands within an individual; that individual is picked.

\*insert pick of roulette selection

One of the advantages is that it is more likely that a fitter individuals is picked as well as still allowing less fit individuals to be picked which may seem counterproductive but it actually helps keep our population which is important (as discussed in the mutation section). However this method also has the disadvantage of struggling to pick the absolute optimum e.g. for values between 0 and 1 it would struggle to pick the more optimum o.999 from the less optimum 0.998 or 0.997.

One of the way we can fix this is to use a non-parametric fitness selection algorithm where in fitness is only measured by “bigger is better”. One method that does this is Tournament selection

###### Standard Crossover

Randomly select one crossover point in each parent and swap the subtree of the node selected. Most Crossovers end up just swapping two leaf nodes, a counter balanced was proposed by Koza (1992) in which a branch node is selected as the crossover point 90% of the time where as leaf nodes are selected the other 10% of the time.

###### Uniform Crossover

As proposed by Poli and Langodon (1998) Uniform Crossover is much like One Point Crossover in that it also traverses a path until in both the parents where the nodes on both the paths share arity, it does this until it comes to a mismatch in arity where the arity of the node in one parent doesn’t match the arity of a node in the same place of the other parent . Essentially, we map put a “common” area between the two parents starting at the root node. Once we have this “common” area; each node in the common area is swapped with uniform probability. Also, if a node is on the edge of the “common” area and it is also a branch node then the subtree below it is also swapped else only the nodes themselves are swapped.

Uniform crossover is considered to be the superior of the three due to it being unbiased and allowing for Global search meaning that a truly optimal program is more likely to be found.

##### Tree Style Genetic Programing Pipeline

Tree style GP is the most common form of genetic programming. It, as the name implies, uses trees as its representation. Usually GP are constructed like GA’s except their crossover and mutation is different in that it does crossover with 2 parents 90% of the time and 10% of the time the parent is directly copied. There isn’t usually mutation as the strange crossover method is usually enough. However, mutation is still possible. This GP normally uses Tournament selection with a tournament size of 7.

# Project Management Plan

Research the multiple different implementations of genetic algorithms

## Work Breakdown Structure

### 1 –Initial Set-up and Research

* 1. - Research multiple different Genetic Algorithm implementations
  2. - Pick one Implementation method for GA’s to start with initially

1.3 - Acquire the use of Webots and learn basics

1.4 - Create first environment with flat ground in Webots

1.5 - Create Initial robot body in Webots

1.6 - Add a start position and goal for Robot

### – Flat Ground Scenario Genetic Algorithm

* 1. – Implement first Genetic Algorithm for flat ground traversal
  2. – Combine the algorithm with the Webots Robot
  3. - Run GA in the Flat Ground scenario until Robot reaches goal or time runs out
  4. - Modify GA until Robot reaches goal or try different method for GA
  5. – Continue to run GA until Robot converges on method to reach goal

### 3 – Uneven Ground Scenario Genetic Algorithm

3.1 - Change Webots environment to include just uneven terrain on path

3.2 – Run Flat Ground GA on Uneven ground until robot reaches goal(Risk: or time runs out)

3.3 – Modify Flat Ground GA until Robot Reaches goal (Risk: or try different method for GA)

3.4 – Continue to run GA until robot converges on method to reach goal

3.5 – Add in flat ground and Uneven ground to simulation

3.6 – Run same GA on the new terrain until robot reaches goal

3.7 - Continue to run GA until Robot converges on method to reach goal

### 4 – Obstacle Scenario Genetic Algorithm

4.1 – Change Environment so there is a single obstacle on path to goal

4.2 – Run Uneven ground GA on Obstacle path until robot reaches goal

4.3 – Continue running GA until Robot converges on method to reach goal

4.4 – Add in flat and uneven ground to simulation before obstacle

4.5 – Run Same GA on new terrain until robot reaches goal

4.6 – Continue to run GA until robot converges on method to reach goal

### 5 – Final Testing of Genetic Algorithm

5.1 – Take final algorithm and run it on flat ground

5.2 -

### Webots

* Implement Robot in Webots
* In depth learning of the functionality and various uses of Webots
* Create alternate bodies in Webots

### Implementing Genetic Algorithms

* Implement first Genetic Algorithm in robot for traversing flat ground
* Use Flat ground GA for uneven ground and modify accordingly
* Use first ground GA for

### Testing

* Test Robot on flat surface
* Track how long Robot takes to learn path for flat ground
* Track how long Robot takes to learn path for uneven ground
* Track how long Robot takes to learn path for Obstacles
* Track how long Robot takes to learn path when robot has to jump over a gap
* Use Webots to Create a suitable simulated environment for the robot to move in ( M )
* Create a body for the robot to operate in Webots ( M )
* Add alternate bodies for the robot to operate in Webots( C )
* Create a genetic algorithm that allows a robot to traverse flat ground over a short distance ( M )
* Create a genetic algorithm for a robot to traverse uneven ground ( S )
* Create a genetic algorithm that allows a robot traverse obstacles ( C )
* Create a genetic algorithm that allows a robot to jump over gaps ( W )
* Apply different approaches to GA’s throughout the different scenarios ( C )
* Optimize the algorithm to move the robot efficiently across the terrain ( W )
* Have the robot learn the path from scratch as quickly as possible ( W )
* Have the robot learn to navigate a multitude of terrains at once ( C )
* Implement an initialisation process that produces a diverse initial population
* Implement a Fitness function which accurately tests each program on how optimal it is and select programs based on this
* Implement genetic operators which produce a suitable and diverse next generation

ss