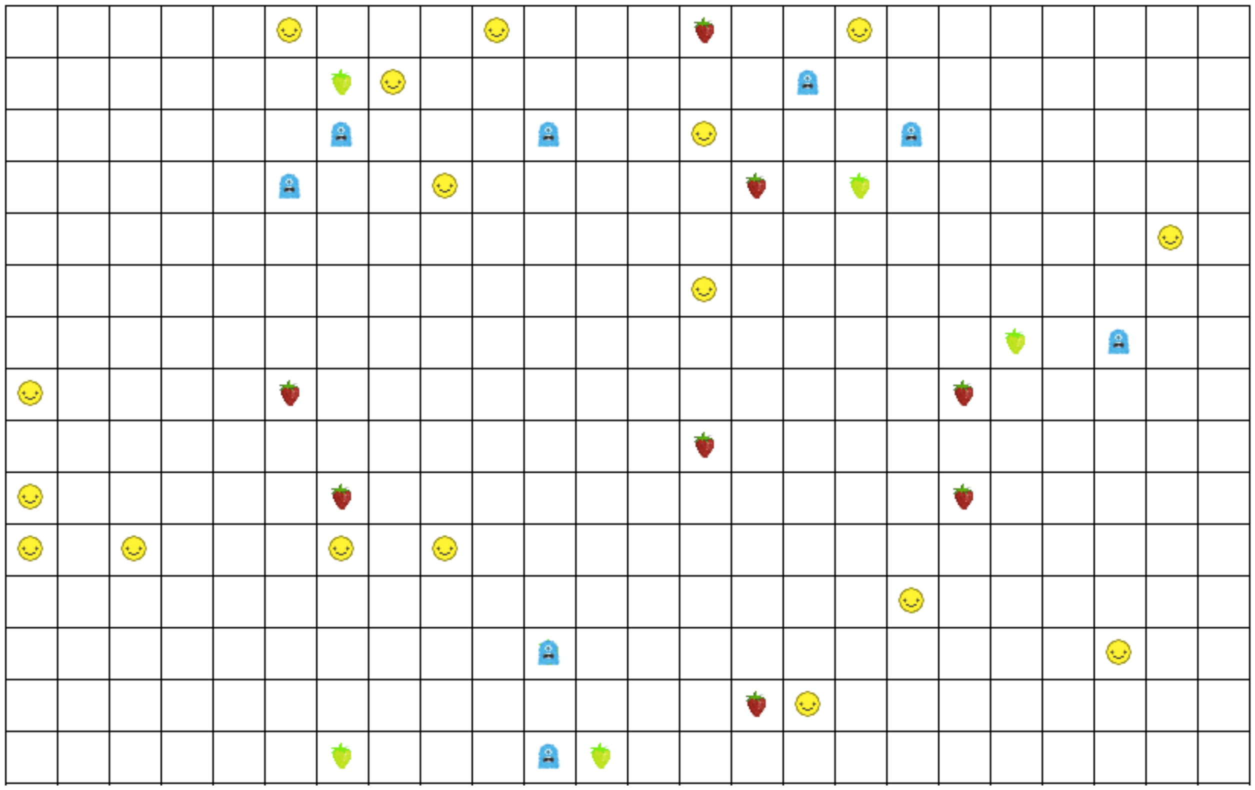
COSC343:

Artificial Intelligence Assignment 2

Tyler Baker 1927791

16/05/17



Abstract:

An experiment to determine the speed and repeatability of genetic algorithms was conducted. Creatures that were programmed with a set of chromosomes to dictate each individual's response to immediate surroundings were run through an engine that presented them food and foes. The initial chromosomes for the first generation were generated randomly, but each subsequent generation descended from the "fittest" of the previous generation. This was to preserve the best possible combination of genetics in each generation. The "fitness" value of the first three and last three generations proved statistically significant, showing rudimental levels of evolution finding optimal genetics.

Introduction:

A genetic algorithm is defined by it's ability to solve unbound optimisation problems using a variation of natural selection, preserving the optimal combinational logic from a previous run or generation. Genetic algorithms are a family of computational models that are inspired by evolution (D Whitley, 1994) and use the fundamental concepts of natural selection, or in this case, survival of the fittest. The fitness of the creatures in the experiment were defined by their ability to survive more turns and maintain higher energy levels. Mutation, defined as the change in structure of a gene, was incorporated into the population. This is an essential component of natural selection, as it is one of the only ways to improve a gene pool. Mutation was incorporated as a random chance for any given creature of a new generation to reroll a single gene. This overwrites inheritance and only applies to a single gene of the creature.

The monsters are a slow moving hazard within the engine. These were implemented to give chase to the creatures, killing the creatures if they catch them (enter the same tile). This is to help create a survival aspect to the simulation, where creatures that are better at avoiding monsters have a higher survival rate than those that do not. The third implementation into the engine was food. This boosted the creatures' energy levels, and was required to survive to the end of the simulation. Food came in two forms: green strawberries, theses would give energy but cause sickness that prevented the creature from moving for two turns, so a risky trade off. The second was red strawberries, these were ripe and the primary food source with no repercussions.

Engine Specifications:

The settings for the engine could be altered in order to see how populations performed on different sized maps and with longer or shorter rounds. For this experiment, a grid size of 24 and a population of 34 creatures was picked. After trials with a grid size of 40 and 50 were picked, the volume of creatures, food and monsters caused the simulation to lag significantly. The population was also so large and the gene pool so diverse relative to the amount of chromosomes, that the second generation was almost fully optimised. A trial with a grid size of 15 made the simulation work very efficiently, but the gene pool was not large enough to show any fitness results, as the only improvements could come from mutation. A default of 100 turns was selected, as this required the creatures to eat multiple times and survive a handful of encounters with monsters. Too few turns did not test the ability for the creatures to eat, and too many turns almost guaranteed that the creatures could not find any food towards the latter of the simulation.

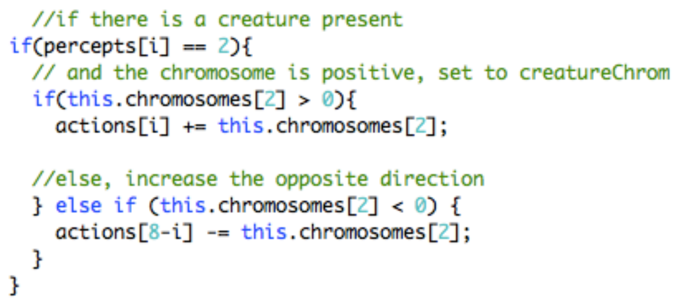
Agent Function and Chromosomes:

Initially, the first generation of creatures was given a random value for each of it's chromosomes. This would be used to derive an action when each chromosome was applied to a percept. There were 6 chromosomes:

chromosomes[0] randomChrom	- the value of performing a completely random move.
chromosomes[1] monsterChrom	- what to do when a monster is spotted
chromosomes[2] creatureChrom	- what to do when another creature is seen
chromosomes[3] foodChrom	- the drive to move towards an unknown type of food
chromosomes[4] greenChrom	- how much a creature wants to eat a green strawberry
chromosomes[5] redChrom	- how much a creature wants to eat a red strawberry

Each of these chromosomes values could range between -5 and 5, with 5 signalling an attraction to the percept and -5 being a repulsion. As each round finished and a new generation was formed, these chromosome values would be passed down from the fittest creature.

The ability for a creature to react to it's immediate environment was governed by an AgentFunction. This mapped chromosome values to a given percept, dictating the outcome of the following move by the highest weighted response. Each action the creature could perform is initially weighted at 0, so if a value of a chromosome is positive then that action would take priority over lower valued actions. The AgentFunction worked in such a way that if a chromosome had a negative value, the opposite movement was increased, simulating running away from the direction the percept was recorded at.



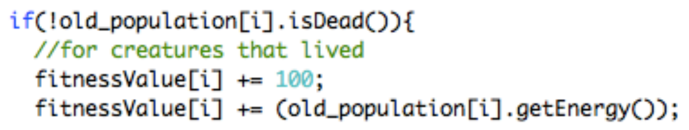
```
  //if there is a creature present
if(percepts[i] == 2){
  // and the chromosome is positive, set to creatureChrom
  if(this.chromosomes[2] > 0){
    actions[i] += this.chromosomes[2];

  //else, increase the opposite direction
  } else if (this.chromosomes[2] < 0) {
    actions[8-i] -= this.chromosomes[2];
  }
}
```

Figure 1.0: A sample of how the creatureChrom influences actions

This method was picked as it is easy to read assess what values are preferable when analysing the final results. This is easy to implement and does not interfere with the nature of the experiment nor hardcode a response to percepts.

Fitness Values and Genetic Algorithm:

Following the end of each round, the generation was assessed on it's performance relative to maximum. Any creature that lived was given 100 points, and points relative to it's remaining energy. This gave a cap of 200 points for any creature that lived the entire round and was fully fed.

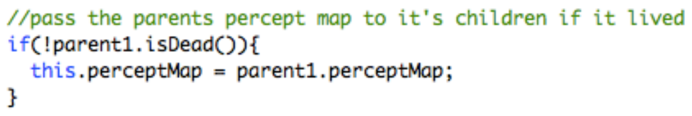


```
if(!old_population[i].isDead()){
  //for creatures that lived
  fitnessValue[i] += 100;
  fitnessValue[i] += (old_population[i].getEnergy());
```

Figure 2.1: A code sample of how living creatures were awarded a fitness vale

If a creature died, it was given a fraction of it's energy remaining. This was because creatures that died immediately did not scale to 50 points (the starting energy) as this would mean the creature was half as fit as another that lived the simulation with very little energy remaining. I initially based fitness values for dead creatures relative to the average turns the population lived, but later based it off the total turns the creature could live. This was because new generations were not achieving higher fitness ratings than ancestors as the whole population evolved.

The genetic algorithm selected the fittest creatures to reproduce. This was done with a sorting method that could store the highest and second highest values of the population. The top two creatures then gave offspring to the next population, with the chromosomes of the two being randomly selected for each new creature. To mimic real sexual reproduction, exactly half of the chromosomes came from each parent, but in a random order. Following this, a unique perceptMap discovered by one of the parents could be passed down to the next generation, as long as the parent was still alive. This simulated a child learning from it's parents. The map was used to determine which square the creature was in relative to it's percepts by way of deduction.



```
//pass the parents percept map to it's children if it lived
if(!parent1.isDead()){
  this.perceptMap = parent1.perceptMap;
}
```

Figure 2.2: A code sample of how living creatures could pass their perceptMap

This map was used to distinguish between the two types of food that the creature could be standing on, and allowed a more specific eating function, much like a child learning what berries to each and which to avoid. The final section of the genetic algorithm, and arguably the most important, was the ability to any chromosome to mutate at random. This provided the population the ability to bring new values into the gene pool.

Results and Discussion:

After running the simulation, a reading of both the average fitness value and survivors of each generation was collected. This was put into a graph to display the evolution of the population.

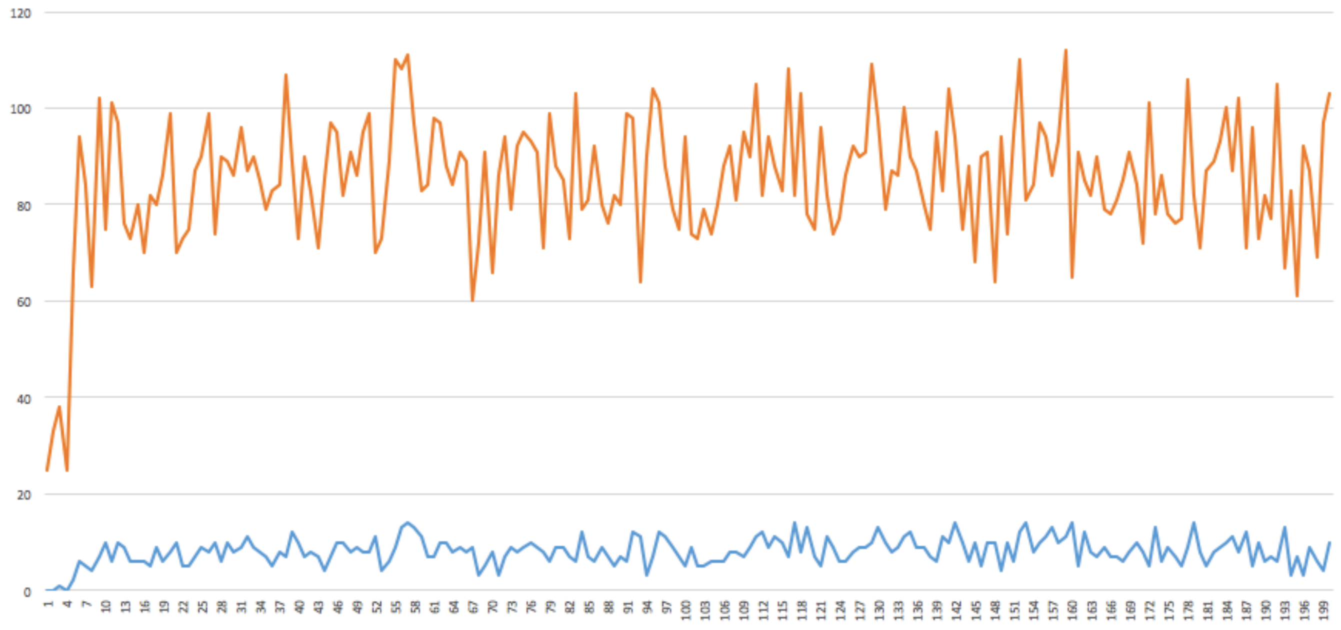


Figure 3.0: A graph of the entire simulation for one population.
In orange is the average fitness value and in blue is the amount of survivors per generation.
X axis: generation number (max 200)

To determine whether the fitness values for the first and last generation was statistically significant, a t-test was performed. Values for the first 5 and last 5 generations were entered and a significant difference was shown, with a P value of 0.00145. The graph shows rapid development following generation 7, this could be due to the narrowing of the gene pool and the rapid decline of less effective creatures. By about generation 30, there is a stable fitness average of 90, never dropping bellow 60 again. I believe this stability comes from the fact that there is always a survivor from the previous generation, and the perceptMap is never lost. In other trials of the simulation, an anomaly where an entire population is wiped out results in several generations of disrupt as the perceptMap is redrawn.

One of the two major discoveries from this experiment was the effect of randomChrom. Initially, it was predicted that randomChrom would be a positive value so that if there was nothing in sight for the creature it would roam until it found food. What happened very commonly was the

opposite, the creatures that had a negative randomChrom were favoured, as this meant the default was for the creature to head to percept[0]. The reasoning behind this was that it kept the creatures going in a single direction, which covered ground more effectively.
The second discovery was that frequently creatures developed a negative creatureChrom, this was because after many generations creatures became very adept and avoiding monsters, and the biggest competition that they faced was for food amongst themselves. Having a negative creatureChrom caused the population to spread out and potentially find a source of food which the creature did not have to share. This was particularly interesting as it goes against the common strategy of safety in numbers.

Conclusion:

This genetic algorithm experiment gave good insight into the logistics of running an unbiased experiment, as well as show how the specific requirements for the evolutionary mechanisms need to be implemented. The results showed significant differences between the initial generations and the fully refined offspring, which gave a good demonstration of how genetic algorithms can be implemented into optimisation problems.

References:

Whitley D (1994). A genetic algorithm tutorial. *Statistics and Computing*; DOI: 10.1007/bf00175354.