COSC 343 Assignment Two
Evolve a Species

RHIANNE PRICE 1953770

The Simulation

- Grid size: 30
- Turns per generation: 100
- Generations: 500

I have chosen a grid size of 30, which leads to 53 creatures per generation. This is a good size because you get a little bit more diversity coming through with a slightly larger population than the default, and it doesn't take too long to run each simulation. I have left the turns per simulation as the default because this seems to be enough to test the fitness of each creature and weed out the unfit. I have decided on 1000 generations. As I will explain later, this is usually long enough for the creatures to evolve to some level of intelligence.

Agent Function and Chromosome Parameters

The model in my agent function uses simple weighted sums to rank the attractiveness of actions available to a creature. Each chromosome has eight preference parameters which determine the relative attractiveness of each action. This function maps the percepts onto actions using the information that the positioning of percepts in format 2 correspond directly to the positioning of the actions.

The first 9 values of the action vector are determined by the following preference parameters:

- preference for moving onto a monsters square
- preference for moving onto a red strawberry
- preference for moving onto a green strawberry
- preference for moving onto another creature's square

For each square, the value of the action vector for that square is the sum of the preferences for each of the things present on that square. For example, if the percept vector indicates that there is a red strawberry and a monster on square i, the value of a_i will be equal to the creatures preference parameter for moving on to a square with a red strawberry plus the creatures preference parameter for moving onto a square with a monster. This results in the creature ranking each square according to its preference parameters what it detects to be present on all the squares. These parameters initially take a random value in the interval (-1, 1), but can grown in either direction through mutation. I have chosen this range because my agent function uses an additive model.

The action corresponding to random exploration is determined by another preference parameter.

There are three extra parameters in the creatures chromosome, which all relate to eating food. The creature's chromosome has a parameter representing its preference for *eating* red strawberries, and a parameter that represents its preference for *eating* green strawberries. Since this percept format doesn't reveal whether there is food on this creature's current square, the creature starts off by guessing an integer in the interval (0, 8) to represent which square it is on. This is the third parameter. If there is food detected on that square, the action that tells the creature to eat takes on the value of the creatures preference parameter for eating the type of food it detects on that square.

This is a very simple model for mapping percepts to actions, and doesn't acknowledge any relationships between different percepts, so doesn't allow for any decision rules based on combinations of different percepts. It also doesn't allow for any "runaway from" behaviour either.

Genetic Algorithm

I have implemented my genetic algorithm using a fitness function, parent selection, some elitism, chromosome crossover and mutation.

FITNESS FUNCTION

The fitness function I have chosen is again a weighted average of the information we have about the creatures. I have three weights in my fitness function which are normalised to sum to one. They determine the relative importance of the following pieces of information:

- The creature's energy level at it's time of death
- The creature life span (=time of death if dead, =numTurns if alive)
- Whether the creature is alive or not

I found placing the most weight on the creature's energy level, then lifespan, then whether it is alive or not is a good ordering. This encourages evolution in the direction of creatures with high energy levels (which implies creatures who are good eaters), long lives (which implies that they are good eaters and averse to monsters), and creatures who live through the whole simulation (which again implies good eaters and monster averse preferences). Changing the relative sizes of the weights has a noticeable effect on the outcome of the evolution simulation. I haves used trial and error to find a good balance.

To encourage creatures to eat I have placed a lot of weight on energy level at time of death. This is good in the earlier generations, but once the creatures become good eaters, they weren't really evolving strong enough (averse) preferences towards monsters. To solve this, I have introduced a multiplicative term involving the creatures energy level if the creature is alive at the end of the simulation.

If a creature is dead, it's final fitness is given by:

$w_{life\text{-}span}\, life\text{-}span + w_{enery}\, energy$

If a creature managed to survive it's fitness is given by:

$w_{life\text{-}span}\, numberOfTurns + w_{enery}\, energy\ (1 + w_{survival}) + w_{survival}$

N.B. This fitness function leads to an average of around 30 before evolution. After evolution, the highest scoring creature in the population usually gets to around 120, and the population usually averages around 60.

PARENT SELECTION

I have used roulette style selection with some elitism to determine which creatures will be the basis for the next generation. I have implemented this by normalising all the creatures fitnesses to sum to one, choosing two random floats in the interval (0, 1], and then arranging the creatures in the order they were generated along this interval according to their normalised fitness score and choosing the creatures whose interval contain the random floats. This means the same creature can be chosen twice. The element of randomness here introduces a sort of controlled diversity to the population, whilst tending to pick the fitter creatures. I initially tried a tournament style method of parent selection where the highest scoring creature from each half of the population were chosen, but it resulted in a level of diversity which stunted the evolution process, even when I turned down the probability of mutation.

Each generation has a bit of elitism as well; every 10th creature out of 53 is a (potentially) mutated copy of the highest scoring creature. This limits the diversity of the population and has the effect of steering the evolution in the direction dictated by the fitness function.

CROSSOVER

I have taken a simple approach with crossover. Given two parent chromosomes, the child chromosome takes the first four parameters from the first parent and the last four from the second parent. I experimented with different combinations and found that taking one parents eating related parameters, and the other parents movement related parameters and combining them lead to the best results.

MUTATION

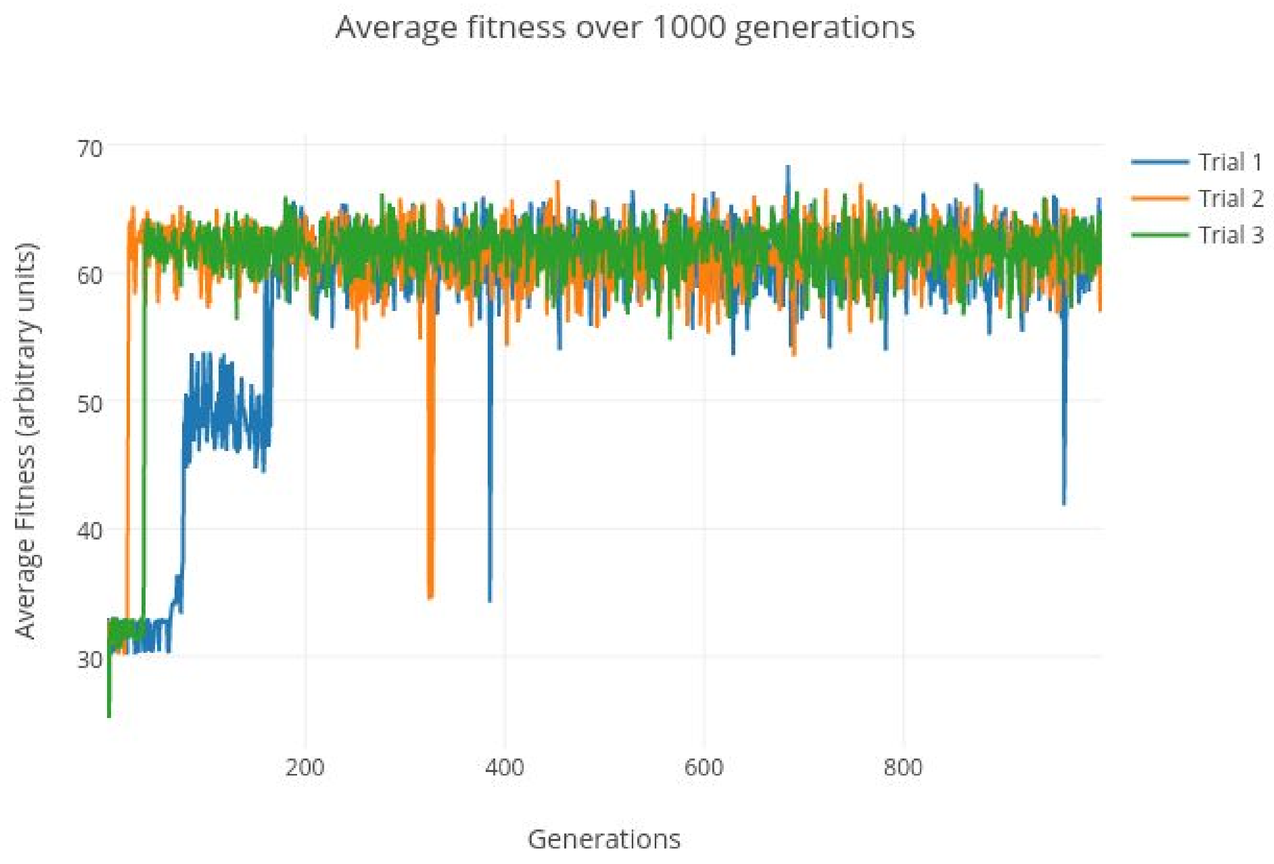
After a new chromosome is made through this crossover process, with a low probability, three of the creatures first six chromosome values are adjusted by a random float in the interval (-1, 1). With an even lower probability, the chromosome makes a new guess at the square it is currently on. Since I have decided to increment (or decrement) parameters by a random float in the same interval each time, the parameters have a lot of room to grow, and can grow quite quickly. This allows for the chromosome to have strong relative preferences, and a concrete and consistent ranking system. If I had limited the range in which these parameters could grow the difference in relative preferences would have been more marginal and may not have worked as well. This comes with a danger though.

In some cases, it takes too long for the creatures to settle on the correct square guess and they don't learn to eat. In the meantime, other parameters can stray too far from zero in the wrong direction to be corrected in later generations. To get around this, instead of only generating creatures from crossover and elitism, some creatures are given a completely fresh set of chromosome parameters. With a low probability, each creature's chromosome could be generated in the same way that the first generations chromosomes are generated. Doing this increases my model's robustness to mistakes in the evolution process.

Discussion of Results

My implementation of the agent function and genetic algorithm results in a level of evolution and some intelligent behaviour. On inspection of the change in chromosomes of later generations, the preference parameters of the fittest creatures tend to ranks actions consistent with the world and maximising average fitness according to my fitness function. Creatures can be seen moving toward and eating food, and even finding green strawberries and waiting for them to turn red before eating them. They mostly avoid monsters, but there is no structure that enables them to run away from monsters.

As the following plot illustrates, over the generations average fitness tends towards a natural level, with some volatility. N.B. please find an HTML version of this plot with 5 trials to illustrate the possible variation in evolution paths,



One might expect to see gradual improvements in average fitness over the generations, but in my implementation I usually see a sudden jump in fitness and little improvement thereafter. This jump usually occurs within the first 500 generations, so I have put the number of generations up to 1000 to try ensure that the jump happens. This jump is a result of the structure of my model and the randomness of the engine. A creature's ability to detect food on it’s square, and it’s propensity to eat that food is the most important determinant of fitness and survival under the model I have chosen. The jump happens when enough creatures guess the correct square, have a high enough preference for eating food to rank eating higher than other actions, and also happen to evade monsters for long enough to survive. When these three things happen to a creature, it is the clear winner of it’s generation and the average fitness of the next generation increases significantly. Very occasionally this jump doesn’t happen in the 1000 generations so I have provided a screen capture of the simulation just in case.

The plot shows these rapid increases, but also some rapid decreases in average fitness, and a lot of general volatility in fitness between generations. The structure of my roulette style parent selection algorithm allows for backwards evolution. Since every creature still has a chance of being chosen as a parent, even after fitness has been determined, unfit creatures may be chosen from time to time. This allows for sudden drops in average fitness to happen. The volatility is a result of the randomness of the world. Sometimes fit creatures die anyway, and sometimes unfit creatures last.

In developing my model I have found that there is a fine line between too much simplicity and over-parameterization. In order to get past the 10 survivor mark, I tried to implement a model that allows for the estimation of relationships between different squares. In theory, this would allow for avoidance of not only squares where monsters are present but also adjacent squares. This worked to an extent, but required 36 extra parameters to be evolved to fit the model. I saw no significant improvement in fitness.

To avoid having too many parameters, and still get this increase in intelligent behaviour, I tried to use the "run-away from" information using the same parameters my already model has. I tried to implement this by not only adding the relevant parameter to the action corresponding to i^{th} square, but also *subtracting* that parameter from the $(8-i)^{th}$ square. This did not improve the model's performance because it clouded the transitivity of the creatures preferences and meant they were less likely to develop strong preferences for eating. To avoid this, I thought instead of using the negating the parameter, I could split each movement parameter into two separate preferences, one for moving toward, and one for moving away. This would only add four extra parameters to the chromosome. This idea didn't result in any increase in average fitness or survivor count, so I left it out of my agent function. My final model is simple but still breeds some intelligent behaviour.

Implementation notes

- Agent function is implemented in the `MyCreature` class.
- Parent selection is implemented in the `new_generation` function in the `MyWorld` class.
- Crossover and mutation are implemented in the constructor in the `MyCreature` class.
- Each creature has it's own random seed.