Genetic Algorithms

Evolve a species Assignment 2 COSC343 | Patrick Baxter, ID:1720748

The assignment requires us to implement a genetic algorithm that optimises the fitness of a creature within a simulated world. It outlines the world as having randomly placed food that gives energy for creatures to move, and monsters that can kill the creatures. The implementation we create has an agent function which the creatures calls to move, and a genetic algorithm which evolves each generation. Our goal is to keep the creatures well fed and alive by optimizing the fitness score which is based on the amount of turns it survived and the energy.

Genetic algorithms rely on variance within a population to make certain decisions. In our case the variance creates some creatures that do well and others that may not. We score based on this and make a new generation with changes being made in our genetic algorithm. The world the creature navigates determines a large part of how well they do and what the resulting fitness score will be. The variables we can change are, *gridSize* (size of the world), *numTurns* (number of turns), *worldType* (choice between world 1 and world 2), and *numGenerations* (number of generations to evolve).

Grid size gives us the space for creatures to move and how many creatures there are, the default is 24. I found doubling the size to 48 develops traits that improve their scores as there is also an increased number of creatures, creating variance. Creatures have enough energy for 50 turns, so having the number of turns less than 50 would only breed creatures that are good at avoiding monsters. I chose 100 turns as it is enough for creatures to eventually reach 100 turns giving bonuses to their fitness score, while also not being too hard as to not reward higher scoring creatures. When testing higher number of generations, I found that after only 100 generations the fitness scores would start to level out based on the graphs, so I limited the number of generations to 100. While all these factors played a role in giving higher scores, the genetic algorithm still works with changes to these numbers and still performs to improve each generation.

Creatures each have a chromosome that is an array of 11 values that help the creature decide what action to take. The actions are also an array of 11 values, the first 9 values determine the direction of movement, 10 is to eat, and 11 to do a random movement. In world 1, the movements are mapped to the same index in the array every simulation, so we can statically code behaviour that helps the creature. In world 2, the movements are randomized between the first 9 indexes. This creates a greater reliance on the genetic algorithm to perform, thus I chose world 2.

Each generation has creatures with variance which stems from the chromosome. The chromosome determines the probability of it doing a certain action. We use percepts in the agent function to map to our chromosome, and based on highest value it makes a decision. Because actions are randomised, all squares are possibly affected by any of the percept. I have an array of 11(number of actions) mapped to four times the number of percepts giving us 36 for each giving us a shape of (11,36). The four times the number of percepts is because I'm utilising one hot encoding for creating a weighted matrix.

When a creature wants to move it calls the agent function. This function gives us percepts which are the information of the squares around the creature. Based on this we make return an array of 11 numbers. Whichever value in the array has the highest value is used as the creature's action. These percepts have 4 possible values at each of the 9 indexes, 3: food, 2: creature/red strawberry, 1: monster/green strawberry, 0: empty/no food. I can not statically code percepts to map to certain actions, so instead I would need to create weights based on what the values are. We remove the

relation of each classification, 3,2,1 representing the percepts, by creating dummy variables with one hot encoding creating an array for 9 x 4 = 36 to allow for binary classification. This array can be multiplied with each index in our chromosome and summed to gives us a single value at each of the 11 indexes.

The genetic algorithm tries to influence improvements for the new generation by selecting the top creatures(elitism), selecting based off fitness score, breeding new creatures with crossover, and finally mutating genes to add variance.

All these decisions were based on fitness score which indicates how well a creature did. I used 3 times the amount of turns plus the energy if the creature had died, and if it survived, I would also give it an extra 120. This bonus was simply added to promote growth for certain strategies.

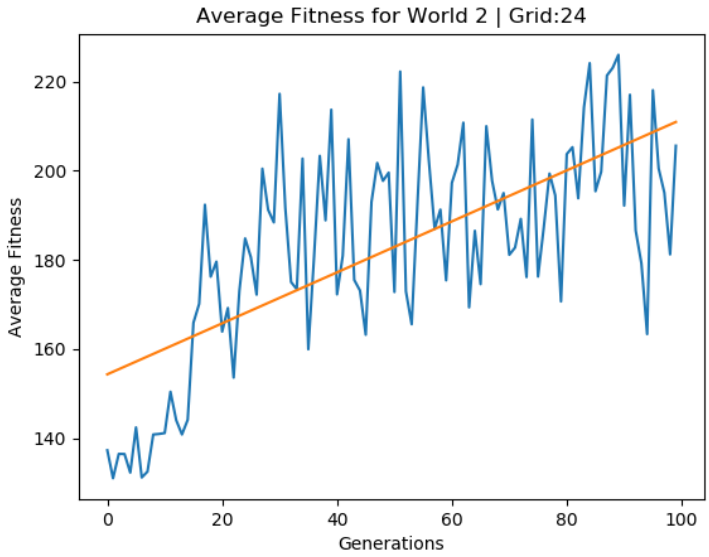
Elitism factored in to stabilize each generation, as without it there was potential for all progress to be lost to random changes. In my case, I took the top 5% of each generation to go on to the next generation.

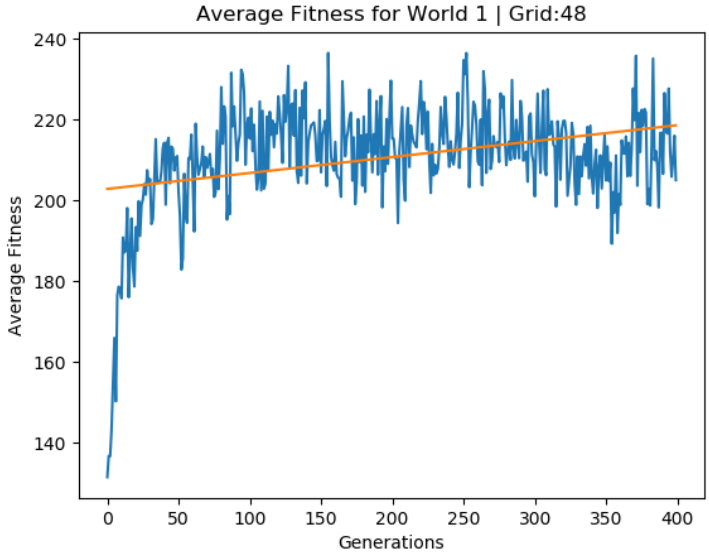
While developing I used roulette selection but I found due to my fitness function, there was little difference between each creature's score, so I instead used tournament selection. We obtain a sample which is the same number of creatures minus the number of elite creatures, by continually selecting 4 random creatures and adding the one with the highest fitness score until you have a full set.

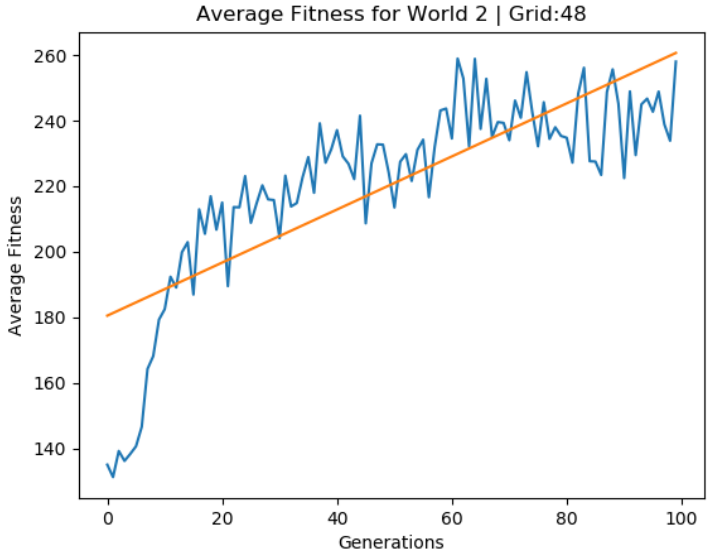
This sample creates new chromosomes using crossover. Crossover takes a point in each parent's chromosome in my case I chose a point within the first 9 indices (as they relate to movement) and two children are created with each side, e.g. P1: (1,2,3,4,5,6|7,8,9…11,10) P2: (9,8,7,6,5,4|3,2,1…10,11) would create C1: (1,2,3,4,5,6,3,2,1) & C2: (9,8,7,6,5,4,7,8,9); the final two genes are chosen randomly to be added to either chromosome.

We create variance within them to remove symmetry, using mutation. This has a 5% chance for each creature to occur. The method selects a random gene to change, and from another random creature copies the gene from the same index to replace it. Finally, we concatenate the newly created creatures with the elite to be passed on to the simulation.

Based on the linear graphs below it is clear that the genetic algorithm influenced the creature's behavior over time. The grid size factored into when the fitness score would cap off, in our case around 100-150 generations, but it made drastic improvements within a short amount of time. Looking at the behavior of the final simulation there are a number of traits that the creature was picked up. The main technique is utilizing diagonal movement, this can be for avoiding monsters or traversing when nothing is around. I believe this gives the creature two advantages, monsters cannot move in a diagonal creating a greater distance, secondly, it gives information on more new squares than if it were to move straight. On higher scoring simulations, creatures would go back and forth on a green strawberry waiting for it to become red. It should be noted that these were not always the techniques learnt but they were common amongst higher scoring simulations in testing. Overall based on the results, the genetic algorithm is able to drastically improve the overall fitness within just 50 generations and even off at 100 generations. Implementing a more thorough fitness function may allow us to better utilise other techniques such as roulette selection, but with the methods I used they worked well together.


Average Fitness for World 2 | Grid:24
Average Fitness
Generations
140
160
180
200
220
0
20
40
60
80
100


Average Fitness for World 1 | Grid:48
Average Fitness
Generations
140
160
180
200
220
240
0
50
100
150
200
250
300
350
400


Average Fitness for World 2 | Grid:48
Average Fitness
Generations
140
160
180
200
220
240
260
0
20
40
60
80
100