Cosc 343

Starshipladdev

May 2019

1 Fitness function

Maximising the fitness function was the primary goal of this project, however, what the fitness function should be quantified as was a key development point. The fitness function would be used to compute the genetic viability of an agent when producing offspring, and so would have to be directly related to how 'evolved' a high-fitness individual appeared. I iterated through several combinations of values to compute fitness, but ended up using each smileys time of death (In turns) + (said smileys energy/4). If the smiley survived, it would automatically get a bonus of 25, a desirable trait.
Time of death + energy proved an inefficient fitness function, as it taught agents to run straight into a monster, as the earlier they got killed, the more energy they had conserved, and it occurred more often that those that ate and survived, since time til Death and energy both cap at 100 but are not equally valuable. However, just testing time until death also proved inefficient, as non-survivor longest survival ties came from agents who never ate as it risked getting eaten. Reproduction options very quickly devolved into a population that had equal time till deaths due to starvation

2 Selection of world attributes

The world attributes my given solution runs in is as follows :

Title Size : 35
*World Type :*2
Generations : 250
Turn Time : 100

A title size of 35 produced the most consistent data. A title size too small didn't allow me to properly observe the scope of available interactions with other objects due to the lack of individual interactions/low agent count, and a world too large seemed to either have too large rates of starvation or survival, depending on the algorithm used. World 2 gave proof that the agents were choosing actions based on percepts through fitness testing,rather than direct hard-coded action to conditions. This makes the genetic learning algorithm more provably robust.

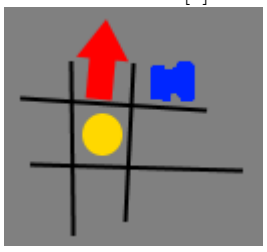
The key thing I wanted with my algorithm was , if not constant improvement,a consistently stable fitness. 100 generations was originally used as my time, as this was when

most test data began to stabilise out, however, I decided 250 generations was both fast enough to not be inconvenient, but long enough that it shows the sustainability of the genetic algorithm used.

A turn time of less than 50-60 meant survivors may not have eaten, so a turn time of 100 seemed appropriate as it allows sufficient time for smileys who's behaviour leads them to starve to be listed as dead,and for smileys to each display their own range of behaviours. However, I didn't want to extend the time too long, as packs of monsters or unfortunate food placement meant as time went on, more smileys would die by chance.

3 Selection of agent Function

The agent function implemented in this assignment was a key component. In all concepts attempted, each smiley, when generated, starts with a random action array, with each index a float between 0 and 0.2.
This was the 'base' agent. Whichever action had the highest value was the 'default' action that the agent would perform if the total effect the agent function had on its actions(see below) was 0. This was an unexpected integral part of the agents survivability and went through a selection and reproduction process equal to the chromosomes.
Originally, each smiley had an array for each percept, and for each of those, an action that it would effect based on which object was in that percept, followed by a value that would effect that action's value.
For example, a Smiley [x], might have the percept [2], where the object is a monster[1], add .5 to action [2].



If this is a generically bad action to perform, the genetic algorithm is responsible for limiting or removing its appearance from the gene pool.
This method seemed robust, however, it became apparent that the agents were prioritizing inconsequential precepts too much. For example, it was a lot harder for an agent to learn to head in an opposite direction to a perceived monster if it could place equal weight on always heading top right. Since the action map was the culmination of all percepts' edits, a chromosome model was selected instead, as it would make prioritizing major influences on the agent's survivability much more acute. The chromosome model is a defined list of several percept and object combinations, such as 'Food on current tile'. Each of these combinations will be referred to as a gene. When an agent is first initialized, each gene is given a random action, with a random weight to add. This is the exact same the percept action-map, instead the agent only reacts to certain percepts.
Weights tested were .2, .4,.8 and 2.
.2 and .4 resulted in a lot of agents simply defaulting to their 'basic' action map,and weights of up to 2 significantly decreased fitness rates due to agents giving too much

value to non-vital actions like moving back and forwards.

Note: Under the parameters of the test, I decided this was the best option, as there are only ever 11 actions in a world containing a set amount of objects. Our focus was on improving the behaviour of the agents in these environmental parameters.Had our goal been to make a more robust agent function capable of adapting to new inputs or actions, I would have kept the method of mapping percepts and objects to random actions to start.

4 Selection of Chromosome Model

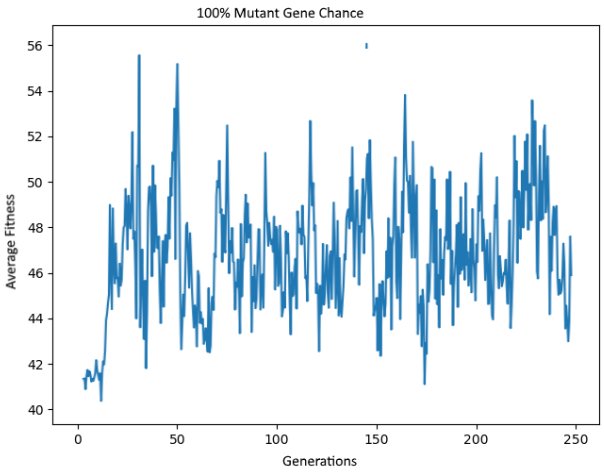
Through various test conditions, the following genes were found to provide the best outcome of fitness.
Each gene has been labeled with its place in the chromosome structure, represented as an array of arrays
Genes with multiple index are separate to each non-agent-square percept (E.G #10 is if food is on north west square, #11 for north)
 # 0 - Chance to do thing if energized
 # 1 - Chance to do thing if un-energized
 # 2-9 - Action (Not weight) if Monster on [x]
 # 10-17 - Action Not weight if Food on [x]
 # 18-Action and weight if currently on Red apple
 # 19- Action and weight if currently on Green Apple
 # 20-27 Action (Not weight) if friend on [x]
 # 28 - Weight for genes 2-9
 # 29 Weight for genes 10-17
 # 30- Action and weight if hungry
 # 31 Action and weight if multiple monsters
 # 32 Action and weight if multiple food
 # 33 Weight for genes 10-17

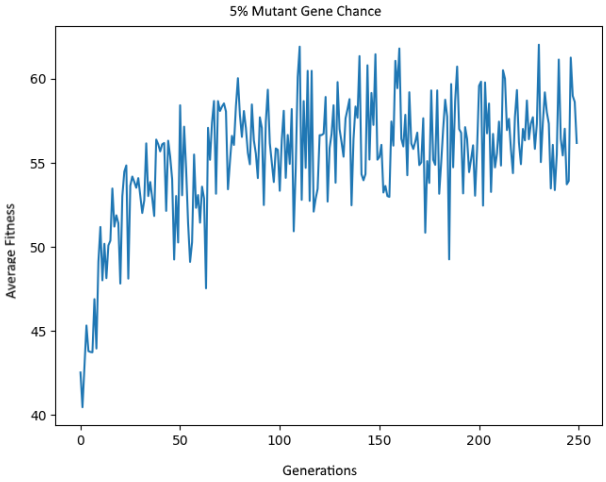
5 Selection of Genetic Algorithm

The basic principle of the genetic learning algorithm followed was to maximize the genes in the population that allowed the longest survivors to live. It needed to account for luck and chance letting otherwise unfit individuals succeed through mutation, and it had to counteract un-fit mutations with genes from the fittest subsection.

Mutation
Mutation was originally I attempted to have offspring be entirely created from their parents genetics, and have mutation implemented via a subsection of the population, 'K', being entirely randomly generated as per the first generation. In this way, mutations would be implemented due to fit 'mutants' breeding with established genes in the population. However, implementing a chance to randomly mutate 'C' amount of genes per offspring provided a slightly more ascending average fitness over time, and

is more in line with the correct implementation of a genetic learning algorithm.
At first, exactly 'C' randomly chosen genes were randomly mutated every generation in each offspring, however, as shown below, making the chance of each gene mutating very low prevented a lot of fluctuation downwards in average fitness. This will be due to most mutations lowering the fitness in an individual, and with high rates of mutation, the fitness will more often than not sharply lower, before stabilising as those unfit genes are removed from the breeding pool.
The following graphs show the difference in averages between simulations that used 100% chance of mutating 'C' amount of genes and 5% chance of mutating each gene.





As can be seen,guaranteed mutant genes, although occasionally providing fitter results, lack the consistency and gradual improvement required. Furthermore, for both tests over an average of 4 runs, The 5% chance of mutation provided better average fitness and survivors per generation.

Parent Selection
Parent selection is done in a tournament style selection process.A subset 'breeding pool' of 'N' individuals from the old population are selected and compared against each other for fitness. The testing method used was changed around to find the best outcomes for the average agent fitness, as is detailed below.
Tournament selection was used to preserve any useful genes available, as 34 genes with weights anywhere from between 0-0.8 require maximizing the recurrence of genetics that increase survivability, as other parent selection methods proved to provide very little improvement in the the agent's fitness.

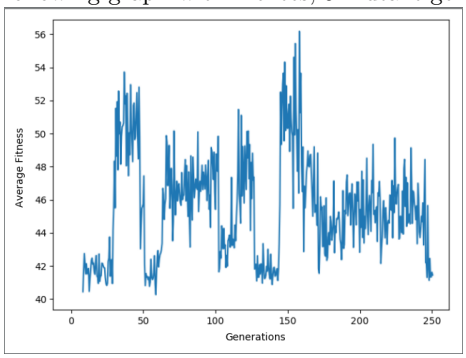
Parent fitness
Although Fitness is modeled as (Energy/4 + Time Till Death), for parent selection the fitness model started as pure survivor numbers. Out of the subsection 'N' breeding pool, 2 random members were chosen as parents before beginning the tournament. Each parent could only be replaced if they died and the competition didn't (Prioritiz-

ing survivability) or the replacement had better timeTillDeath.This was done under the assumption that survivors were obviously fitter, however due to some un-fit individuals surviving via chance,implementing the 'fittest' 2 parents in a subset being the 2 with the best fitness function score(Energy/4 + timeTillDeath +25 Survivor bonus) actually yielded better survival rates and fitness. Again, like the mutation implementation, it was also the genetic-algorithm appropriate way to test parents fitness.
The following table is averaged result of 5 tests, with the same parameters, the only difference being the parent selection function.
Test 1 was with 5 mutant genes, 5 elites and breeding pool of 20
Test 2 was with 9 mutant genes, 5 elites and breeding pool of 20

Test	Average fitness	Average survivors
Fitness Test 1	54	8.7
Survivor Parent 1	45	2.1
Fitness Test 2	53	8.62
Survivor Parent 2	47	2.18

The actual reproduction is done with a *uniform crossover* function, that gives each parent a 50% chance of copying each gene into the new agent's chromosome. Each agents' 'default action' action-map is also copied into the offspring via *uniform crossover*, meaning a 'fittest' default behaviour also emerges after successive generations. Unlike chromosomes, mutations in the default action map crossover gave the best average fitness as 'C'=0. This appears to be non-mutant default actions defaulting to diagonal movement more often (That monsters can't catch), with any other default action due to mutation being inefficient. Due to the start population of 54, this appears to always occur in a gene-set that is carried on, were the population smaller it wold be worth changing to increase the chance of diagonal movement defaults.

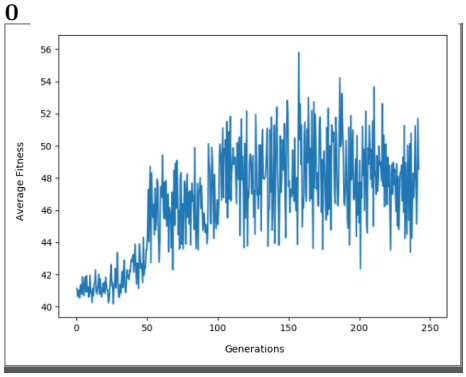
Elites As stated above, genetics that improve fitness or survivability are rare in comparison to the standard random genes, and so require conservation. To further this, an element of elitism is used. A constant 'E' amount of the fittest individuals in the old population are copied in full into the new population. The use of the latest 'elite' as a parent for all of the next generation (Other than mutants) resulted in the following graph with 2 elites, 9 mutant genes ,breeding pool of 20:



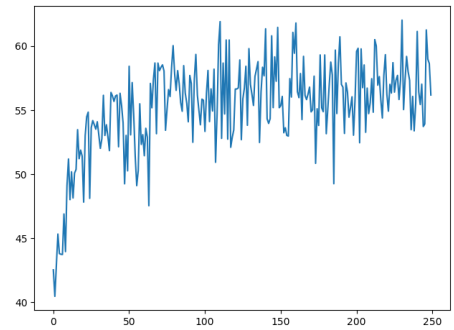
As can be seen, compared to the non-elite parent versions below, this is obviously an inconsistent and eventually worse way to implement genetic learning. This is probably due to the elites' un-fit genes remaining in future generations as well, and by mixing with un-fit parent2 genes as well, they produce unfit offspring.

Theoretically, a large amount of elitism would be a good thing as it would keep survivors' fit genes in the generations,and if genuinely fit, would increase the amount they reproduced those fit genes' action maps. However, after testing to confirm this, it was found that only a small amount of elitism made a large increase to the average fitness of a test population, while too high an elitism value led to poor and inconsistent performance, as shown below.
The following graphs are the averages of 3 runs, with the only difference between conditions being an elitism 'E' of 0,5 and 10

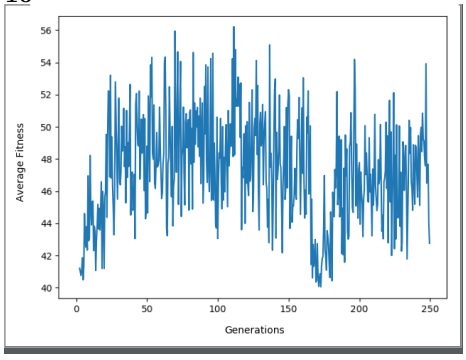
0



5



10



This shows that, as elitism ideally leaves a disproportionate amount of fit genes in

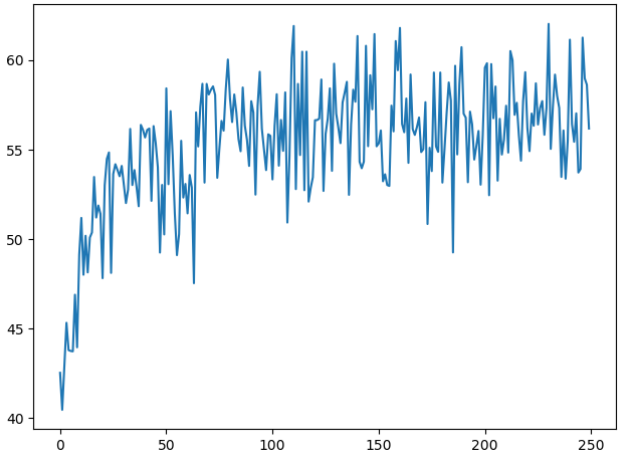
the next generation, if 'E' is too large, there will be a disproportionate amount of sub-par genes carried on as larger amounts of an unfit population need to be kept to fill up 'E' elite spots.

6 Results of Final Selection

Final Selection was a matter of figuring out the best combination of 'K' mutants, 'E' elites and 'N' breeding pool for tournament selection. The end configuration was chosen above others tested, for primarily two reasons:
it produced explainable and intelligent behaviour
it produced repeatable data
A huge part of the assignment was the ability to explain the behaviours that evolved, and the process that evolved them.
Another important aspect to consider was the reputability of the project, and maximizing the evolution explained above.
The following configuration is the supplied assignment's configurations:

large*Title Size :* 24
Generations : 500
Turn Time : 100
large*'K' Mutants Per Generation :* 5
'E' Elites Per Generation : 5
'N' Breeding Sub-selection : 15

After running this configuration numerous times, the fitness total per generation graph produced remains similar. One is provided below as example



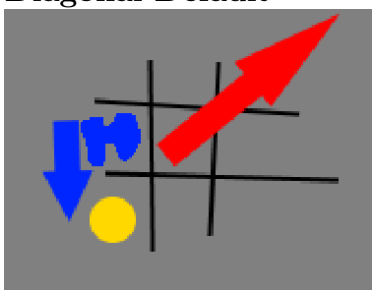
As can be seen, the fitness begins extremely low, due to the randomized genes of the initial population. This is followed by a sharp rise up in fitness that fluctuates around a stable 'point'(aprox. start fitness x 3).
After roughly 60 generations, there is another sharp rise up in fitness (Again, aprox. first rise point x 3), that remains stable and slowly rising around another 'point'.
This can be seen as the genetic learning algorithm finding the right 'combination' of genes between two parents. As the two fittest individuals are included every time,there is a 2/total-pop chance each reproduction that the elite, on top of being reproduced in full, will reproduce with another member of 'N' breeding group and have any flawed or un-fit genes replaced by the other parents superior genes via *uniform crossover*.
Assumedly,this removes a significant flaw in the prior elite, held in a low amount of genes, that makes the new offspring the winner of subsequent tournaments, increasing the fit genes and overall fitness of the population.

It can be assumed using induction, this 'sudden rising fitness' followed by steady slow rises will continue after successive generations even further until a cap due to non-genetic reasons is reached(Not enough fooe for example) .
The fluctuations are a combination of random chance and the mutations, and their offspring, being totally random, and therefore averaging about as fit as generation 0. However, since un-fit genes are unlikely to win tournament selection, the average fitness resets to the

'point'. The same is true in the opposite direction, with successfully 'fit' mutations having good genes overwritten, however tournament selection means the trend will go upwards, especially when the aforementioned 'spikes' occur with two fit chromosome agents producing offspring with the right crossover combination.

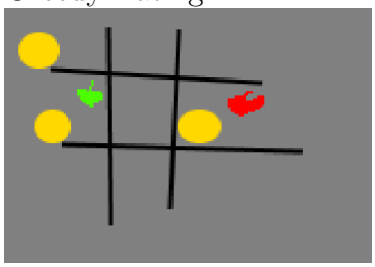
Behaviours Observed The behaviours of the agents with the above configurations at generation 250 are very interesting in terms of how they've 'learnt' some core rules of the environment.
Key and interesting behaviour is listed below

Diagonal Default



This is where the majority of agents move by default in a diagonal line.Although agents appear to react differently to monsters, they do so with diagonal movement. The default movement diagonally will be due to agent's being fitter (Surviving monsters) if they move diagonally to avoid them. As percepts only increment the default action map, that is also generated and passed on like chromosomes, the action-maps that are weighted towards diagonal movement are more likely to be passed on.

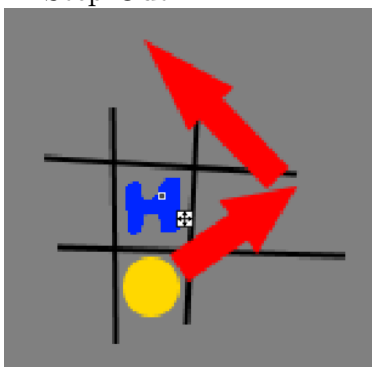
Greedy Eating



An interesting behaviour that evolved in some tests, due to genes 20-27, was that some agents would wait for food to ripen. Although

this is interesting in itself, meaning the survival rates of constantly 'sick' agents would have to be quite low, observation shows agents will sometimes eat un-ripe food if they perceive another agent nearby ad no monster. This shows fit populations have higher weights for monsters than eating, and higher weights for eating ripe food than unripe food. This is one of the more successful behaviours exhibited by some agents as it matches the logical conclusions one could hardcore to produce maximum survival rates. Although greedily eating will not improve the overall fitness of the population, as the disenfranchised agent may starve, it does improve the energy and survival time of the greedy individual, a key requirement of the fitness function

Step Out



A large portion of the final population would avoid monster they come into contact with by moving diagonally around the monster. As stated above, this would have resulted from agents that moved in up and across motions dying apoun early monster contact and having a lesser fitness values to compete in the tournament selection with. As such, most final generation agents 'known' to move diagonally around monsters.