Tom Hodgson

P2420642

Abstract

A modelled simulation of evolution within a rock pool environment; utilizing the principles of Genetic Algorithms and Evolutionary Programming.

CTEC 3451 – Final year development project

Modelled Simulation of Evolution

# **TABLE OF CONTENTS**

***INTRODUCTION***

The Product.………………….………………….………………….………………….……….……….pg03

Products Main Functions………………….………………….………………….…………………pg03

The Products Usage.…………….………………….………………….……………………….…….pg04

Academic Objectives...….………………….………………….………………….…………………pg04

***MAIN BODY***

***Initial Designs, Functionality & Mechanics***

Environmental Creation ………………….………………….………………….…………………..pg05

Creature Creation ……….………………….………………….………………….….……………….pg05

Fitness Tests …….………………….………………….………………….………………….………….pg06

FITNESS TESTING: Fitness Tests between Creature and Environment………….pg08

Population Creation ………………….………………….………………….………………….……..pg10

Mutation Mechanic ……….………………….………………….………………….………………..pg11

MUTATION TESTING: Single Parent ‘Bacteria’ Reproduction Approach…….…pg13

Selection Mechanic ……….………………….…………………….………………….………………pg16

Crossover Mechanic ……….………………….………………….………………….…………….….pg19

CROSSOVER TESTING: Two Parent Reproduction Approach………..…………….…pg21

***Mid-Development Re-Designs & Mechanics***

Reflections & Next Stage Planning………………..……….………………….……………..….pg22

Species Mechanic…………….………………….………………….………………….…………….….pg23

Gene Stacks Mechanic …….………………….………………….………………….………………..pg24

Environmental Capacity Mechanic……………….………………….………………….……….pg24

Creature Weight Mechanic ………….………………….…………….………………….…………pg25

Creature Size Mechanic …….………………….………………….………………….………………pg25

Environmental Status Mechanic…………….………………….………………….……………...pg26

Creature Competition Mechanic………………….………………….………………….………...pg27

Creature Life Span Mechanic ……….………………….………………….……….……………….pg29

Creature Litter Size Mechanic ……….………………….………………………….……………….pg29

User Display………………………….……….………………….………………………….……………….pg30

Fuzzy Logic…………………………………….………………….………………………….……………….pg31

***CRITICAL REFLECTIONS***

Production Evaluation….………………….………………….………………….…………………….pg33

Potential Extensions to Product……….………………….………………….…………………….pg37

Approach Evaluation…….………………….………………….………………….…………………….pg39

Final Thoughts……………….………………….………………….………………….……………………pg39

***BIBLIOGRAPHY***……………….………………….………………….………………….………………………….pg40

***APPENDICES***……………….………………….………………….………………….……………………………….pg43

**INTRODUCTION**

***What is the product?***

The product is a small-scale simulation of evolution within a rock pool environment, developed with the principles of Genetic Algorithms (GAs) and Evolutional Programming (EPs) to simulate creature evolution within an environment.

GAs’ are often implemented to find the best solution for a problem from a mass of random programs; over generations combining and evolving their programs until eventually the best program solution is generated for the problem at hand.

In this project, the problem was survival within a rockpool environment, the solutions were the creature programs; the tests, their ability to survive within the environment. Survival was based upon having the necessary creature variables that match the environmental variables.

Throughout, the program made use of several key GA concepts; including Fitness Tests, Selection, Crossover and Mutation.

Fitness Tests are a concept in which each creature program (referred to as creatures) was assigned a threshold score dependant on their variables, that was used to determine the creature’s success at surviving in the simulated environment.

The Selection process is a concept in which a portion of the surviving creatures are selected to be ‘parents’ and produce the next generation of creatures. Through the Crossover process; selected parents are paired, programs combined and the resulting new programs the source of the next generation. Finally, the Mutation process; determines whether the newly created creature variables will be altered.

Where this project differed to many in its usage of GAs’, was its end goal. GAs’ are often used to find the best solution to a problem, taking many solutions and reducing them to one. This projects intention was to go from few programs to many, resulting in a variety of possible solutions. A simple simulation of natural selection and biological evolution.

***What are the main functions that it performs?***

The main function of the product is to produce an ever-changing population of creatures within the environment over a number of life cycles.

Starting with the generation of a first population from an initial seed population. Over an iteration of life cycles, encourage the populations to increase in numbers and evolve their variable values (their Gene Stacks), to better suit the environment and increase their surviving chances.

The user can iterate through each life cycle, view the changes in the existing creatures’ numbers, their genetic makeup and the changes in that makeup; making comparison between the possible strategies.

***Why is this product needed?***

The product was designed and intended for two principal usages.

Firstly, the program would be used as the core mechanic for a ‘God’ genre game, where the user plays the part of a deity within the game environment, observing and influencing an independent system.

Secondly, to be used as a tool to automatically build eco-systems in games developed by other programmers. An example, game development in which there would be a variety of eco-systems, for instance, No Man’s Skies. This program could be used by developers to automate the process of developing numerous different eco-systems and creatures. Offers the benefits of greater gameplay variety for their game players, as well as being a vital time saving tool during the development process.

The main difference between how these two different use scenarios is how and when the user interacts with the program.

As the base mechanic of a stand-alone ‘god’ genre game, the user would be engaged throughout the program. Whilst as a developers’ tool, the user would be engaged at the beginning when inputting constraints and at the end, when being given the resultant creatures and population.

***Academic objectives of project, what I plan to get out of doing this?***

Foremost was displaying a developed understanding and strong grounding in C++.

Exploration of AI principles such as Genetic Algorithms, Evolutionary Programming and the use of Fuzzy Logic, alongside their use in game development.

The development of automation tools for others. As well as exploring behind the fundamental thinking and considerations when developing for the simulation genre.

Finally, a better understanding of managing and developing a project over an extended time; planning involved, adaptation required and working towards a hard-set deadline.

**MAIN BODY**

***Initial Mechanics***

The key principle of GAs’ are to test multiple solutions to a problem, “GA simulate… the concept of survival of the fittest to guide the search towards an optimal solution over generations of populations”*[[1]](#footnote-1)* (Gunturu et al, 2017). Therefore, a problem, a set of solutions and a test were required; the Environment was the problem, the Creatures were the solutions and the Fitness Tests used to determine whether the solutions were successful at solving the problem.

These three key mechanics were developed in tandem, as they were intertwined and all three would require similar variables to tested against one another. The key survival variables chosen were Oxygen, Temperature and Energy; inspired by the nature that this program was attempting to simulate. If a Creatures’ value for any of these did not meet the environmental variables for energy, temperature or oxygen, the Creature would fail the fitness tests and be considered as failing to survive.

***Environment Creation***

First task, develop a problem that needed solving, an environment to survive. Survival would be tested on energy, temperature and oxygen; therefore, the environment required a variable for each and functions to access these.

Initially, variables were hard set allowing ease of control, for instance creating an ‘Eden’ environment compared to a ‘Freezing Desert’; useful for testing. Later in development, class EnvironmentCreation was implemented alongside the environmentCreation(), randomising and automating this process [*APPENDIX:C01.01*].

This approach was taken as struct Environment was extendable for later developed mechanics [*APPENDIX:C01.02*]; these eventually included:

* Pollution.
* Mutation modifiers.
* Environmental capacity.
* Population weight.
* Capacity multipliers.

***Creature Creation***

Purpose of creature creation mechanic was to build a system which would create a randomised Creature with fully initialised variables. A set of starting point variables were required and a function which would use these to create a randomised Creature.

Struct CreatureSettings was created; contained the building information for each of the core variables the Creatures would require [*APPENDIX:C02.01*]. The idea was to have variables that set the centre point and then others which built a range around that point.

CreatureCreation class was created, key function being creatureCreation() [*APPENDIX:C02.02*]. This function took the above-mentioned setting values, creating a new Creature with its own unique randomised values. Randomising was achieved by a set of randomising functions in the class GeneralFunctions [*APPENDIX:C02.03*], for integers and floats, randomising either in a uniform or gaussian manner.

The main concern in design was extendibility; it was likely that the Creatures would require new variables as the product developed. The approach taken in CreatureSettings and creatureCreation() allowed extensions to be implemented quickly and easily. Over development the following variables/mechanics were added:

* Tolerated ranges.
* Energy multipliers.
* Final demands.
* Litter sizes.
* Life spans.
* Creature size.
* Gene stacks.

These were done so with the minimum of time and effort, which was highly valuable. The CreatureSettings had the additional benefit of being easy to understand, making tuning simpler, saving valuable time.

Within the creatureCreation() there were two methods of randomisation, uniform and gaussian approach.

Centre values were randomised with the gaussian approach as this would increase the chance of Creatures generating more centralised survivable values and would also produce rarer outliers, specialised for more extreme environments.

Range values were randomised with a uniform approach, constrained with min/max.

Efforts were taken for key variables to always have a positive value; it was possible when applying the ranges, for a minimum range to fall below 0.0f. A problem as it would not be possible to survive on negative oxygen and only a few extremophile creatures can survive being -0c. A check and new function resetVariable() were developed to correct this issue [*APPENDIX:C02.04*].

***Fitness Tests***

There were three key tests that a creature must pass to survive the environment; oxygen, temperature and energy. Each tested slightly differently, made sense for three separate fitness tests, inheriting from parent fitnessFunction.

*Energy Test*

A simple test that takes creatures energy demand, compares to environments energy available, passes a bool back, true if passed test (demand < provided), false if it fails [*APPENDIX:C03.01*].

*Temperature Test*

Temperature fitness test a little more complicated, includes an inRangeCheck() and multiplier() alongside the fitnessTest() [*APPENDIX:C03.02*]. The inRangeCheck tested whether environments temperature was in range, ideal and tolerated. If tolerated, then a 1.5f multiplier applied to the creatures’ energy demand, simulating nature, more energy expended to either warm up or cool down.

Bools for ideal and tolerated were then passed to main fitnessTest for temperature, both false then creature fails the test, if ideal true then passes, if tolerated true then passes with a multiplier to energy demands.

*Oxygen Test*

Oxygen test followed a similar pattern to the temperature test but with differing parameters [*APPENDIX:C03.03*]. Tested against the environmental oxygenation rate in the inRangeCheck() and then passed to the oxygen fitnessTest(). If in the ideal range then returned true, if in tolerated range then true with a 1.5f multiplier to final energy demands and if not in either, then it failed the fitness test.

*Extension to Tolerated Range*

First extension to the program was tolerated ranges; extension around the ideal range that allowed survival at the cost of extra energy demand; consider a creature expending energy to warm or cool.

This encouraged creatures to evolve the oxygen and temperature variables as the ‘survived-or-died’ method didn’t; this incentivised change.

Further development; rather than 50% multiplier, adopt a gradient multiplier that increases/decreases dependant on distance into tolerated ranges. Evolution is often a series of steps; cliff-edge drops cause extinctions; gradient effect would offer encouragement to evolve.

*Combined Test*

Final step combined these into a single fitness test, considered a better approach with single call.

Important consideration was the order; throughout development this product has been inspired by the nature of what it’s attempting to simulate, this was no different.

The order was determined by the speed in which each test would kill a creature in real-life condition. Therefore, first was the oxygen, then temperature and finally, energy, as ‘starving’ would take the longest period to perish [*APPENDIX:C03.04*].

Oxygen & Temperature:

1. Checks’ isAlive bool
2. inRangeCheck on ideal conditions.
3. inRangeCheck() on tolerated conditions.
4. fitnessTest() to see if creature survives.

Final Energy:

1. Checks’ isAlive bool
2. Checks’ oxygen & temperature multipliers
3. fitnessTest() on final energy demand against environmental energy provided.

Benefit of approach was oxygen and temperature tests had the opportunity to result in a multiplier to the final energy demands, so the energy had to be last in the order.

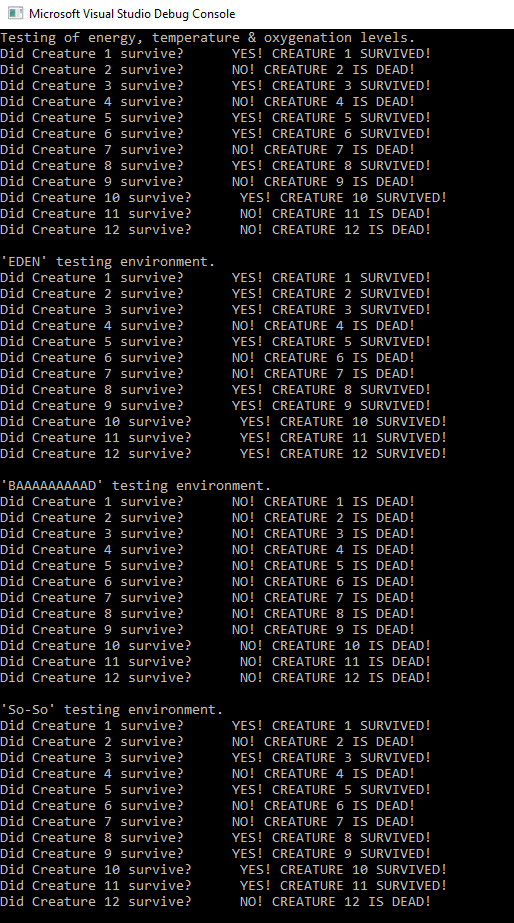
Again, this is easily extendable, for others fitness tests to be included as well as other variables; for instance, the creatures’ threshold score was easily implemented at the end of this function.

***TESTING: Fitness Tests - Creature & Environment.***

The issue with testing randomly created environments against randomly created creatures is that there are no definite expected outcomes; difficult to determine success of implemented mechanics.

Therefore, approach was taken to hard-set a group of 12 creatures and test then against a hard-set group of 5 environments, all with known values. Results are expected and were used to determine the success of mechanics. First, each fitness function was tested separately; then testing all creature against three environments.

Results of survival displayed to the console to be compared with the expected results.



*Fig012.Fitness Function Testing Results – Console Output*

Testing highlighted a problem as some creatures survived which were expected to perish. It became apparent that ranges could fall into negative values. A negative check and resetVariable() were implemented [*APPENDIX:C02.02&C02.04*] to fix.

***Population Creation***

Now the product had a problem, set of solutions and tests; next step was to create populations that the program required to encourage evolution; a set of first populations were needed.

*Generating FIRST population from SEED population.*

Approach decided upon used a Seed Population to generate a First Population, using GAs in a traditional manner; i.e., many random programs to few best programs.

Order of implementation:

1. Large seed population initialised at compile-time in an array.
2. Each creature in seed array ran through creatureCreation(), creating random creatures.
3. Each assigned creature ID with genFuncs->createNewCreatureID() [*APPENDIX:C04.01*].
4. Each creature ran through full fitness tests.
5. Surviving creatures duplicated n amount to build first populations.
6. Creatures passed into First Population.

After testing, value of 15000 set for pool size of seed population. Lesser value result in greater chance of no surviving creatures, larger value provided little benefit over 15000; chance of 0 survival from seed population diminished only slightly for a larger computational cost.

*APPENDIX:B01.02 – Seed Population Pool Size Testing – Failure % - Ran 1000 times on each.*

A check-and-repeat mechanic implemented in case there of zero survivors to ensure first population had creatures.

Extendibility a consideration; later extensions include:

1. Gene Stack mechanic.
2. Species mechanic, surviving creatures start each species.
3. Low intensity mutation of duplicated initial creature populations, gave variety from the initial creature.
4. Extended size of starting populations.
5. Check at least a creature survived from seed population; if not, repeat.

At this point in development, product had a problem to solve, number of solutions to solve it and a series of tests to check their ability to do. The key GA components of Mutation, Selection and Crossover were implemented next.

***Mutation Mechanic***

Mutation is an essential part of GAs, just as it is in evolution. There are two main parts to mutation; deciding whether a variable should mutate and by what intensity. Prior research, “*Mutation is applied with a very low probability*”[[2]](#footnote-2) (Martinez-Arellano et al, 2017), and testing informed decision to ensure mutation chance and intensity values were low.

Mutation class implemented with two key functions, mutationTest() and mutationIntensity() [*APPENDIX:C05.01*]. Called together, first mutationTest(), if returned true then mutationIntensity(), which takes the variable to change and applies a change. A random uniform number was generated in mutationTest(), if lower than mutationChance then attribute mutated.

A gaussian random number generated in mutationIntensity(); gaussian approach taken as it mainly generated smaller mutations but had potential to generate much larger outlier mutation; as shown in the next two tables.

*APPENDIX:B03.04 – Mutation Gaussian vs Uniform – Largest single mutation value*

*APPENDIX:B03.05 – Mutation Gaussian vs Uniform - Averaged Mutation Value*

A benefit because if there were too many large mutations, then chances of creatures suddenly becoming unsuitable for their environment greatly increased.

Potential positive to a rare large mutation was that it could cause a struggling creature to become more suitable to the environment, shifting the species to a more positive evolutional direction.

Tuning of variables took time and experimentation to generate results that felt more suitable; there were no definite results to be targeted in this testing, ‘feel’ was important.

***TESTING: ‘Bacteria’ Approach***

It had been planned pre-implementation for a limited testing of the product at this point. Inspired by ‘bacteria’ which reproduce asexually, a parent divides into two clone creatures, with an opportunity for mutation at that divide. There were all the mechanics in place to replicate this as Selection and Crossover were not yet required.

Benefit of this approach was that it ensured targeted testing of Mutation mechanic; it was understood that mutations could potentially have very large impacts on creatures and populations “*a very small percentage (e.g 1%) of the selected individuals are mutated in a random way*”[[3]](#footnote-3) (Koza, 1999).

It was important to have certainty that any changes were the sole cause of the mutation mechanic. Data on the number of mutations, overall averaged strength of the mutations and the largest mutations were recorded [*APPENDIX:C06.01*].

The plan used the first population creatures, ran fitness tests, if survived, divided into two, repeat for 30 cycles; after cycle 25, the program crashed.

On inspection there were too many creatures after 25 cycles; a population explosion due to exponential growth. Starting populations of 12 for each species would lead to over 400 million creatures per species after 25 turns. This was further exacerbated when debugging showed that the parent creatures were not being removed after ‘birthing’ their two clones.

*APPENDIX:B05.01 – ‘Bacteria’ Testing – Test Set 1 – Exponential Growth*

Another functionality was for the product to produce creatures that experience different population growths due to their unique genetic makeup. At first this appeared to be the case:

*APPENDIX:B05.02 – ‘Bacteria’ Testing – Test Set 2 – Misleading figures suggesting variety*

Despite the program crashing population explosions, it appeared that the product was still producing different strategies to cope with the environmental problems, as there were species with very different membership sizes.

But repeated tests did not replicate these results, instead they all followed a very similar population explosion.

*APPENDIX:B05.05 – ‘Bacteria’ Testing – Test Set 3 – more typical results*

Looking into the Test 2 in more detail, it became apparent that there weren’t different strategies.

Instead, with CreatureID:2760 (lowest population in Test 2), its energy demand was very close the environmental energy provided. Any mutation that increased the energy demand led to fitness test failures, a much lower population and the misleading appearance of diversity in populations.

Looking at the n% of fitness test failures in the populations highlighted this:

*APPENDIX:B05.04 – ‘Bacteria’ Testing – Test Set 2 - % of populations failing fitness tests*

These tests were designed to check whether the mutation mechanic was working and showed the results of tuning. Whilst it did this, it also highlighted some issues and encouraged thought on the necessary mechanics required; including:

* Remove infinite resources within the environment to prevent infinite growth.
* Requirement of life spans, ensuring creatures are removed.
* Greater intelligence required to allow alternate results to population death or explosion.
* Sustainability required.

Before the implementation of these mechanics, it was decided that the key GA components of Selection and Crossover would be implemented as these would be required for the planned two parent model tests.

***Selection Mechanic***

Selection is the process where the ‘parent’ creatures are selected from the existing population, combined to produce the next generation of creatures. Main considerations:

* % of population selected as parents?
* Which would be selected? Threshold score to determine?

Research had suggested three main approaches, “*the parent is chosen using roulette wheel selection*”[[4]](#footnote-4) (Kholimi et al, 2018), “*We used an elitist selection method*”[[5]](#footnote-5) (Cole et al, 2004) and [this study uses] “*using tournament selection*”[[6]](#footnote-6) (Martinez-Arellano et al, 2016); with a fourth approach considered:

* *Roulette* – randomly select n% of population.
  + Pros: no threshold score.
  + Cons: no intelligence.
* *Elitism* – select the top n% of population.
  + Pros: only those best equipped at survival in the environment reproduce.
  + Cons: a threshold score required; populations organised by this score.
* *Versus Tournament* – population split into groups, creatures versus one other, highest threshold score wins, moves onto next round, repeat.
  + Pros: allows for alternate creature approaches within a species to reproduce, potential for greater variety.
  + Cons: threshold score required, process of grouping and versus entire populations, including those with the lowest threshold scores, computationally expensive, lengthier implementation.
* *Combination* – top n% selected and then these creatures versus each other.
  + Pros: reduces computational expense of versus approach as lowest scores removed as were never going to succeed.
  + Cons: large populations needed as two cuts of potential parents.

Roulette was considered to random for this system, Tournament required larger starting populations to be effective; Elitism was the approach adopted.

The next consideration was the implementation of a threshold score. It was decided that the difference between the creatures’ energy demands and the environmental energy available would be this score as it represented the creatures’ excess available energy that could be used to successfully reproduce. This was generated and updated every time the full fitness test was called upon the creature [*APPENDIX:C03.04*].

Finally, n% of the population to reproduce had to be tuned to find the correct balance, allowing sustainable population growth; too small and populations crashed, too large and populations exploded. It was during this tuning process that a potential issue with the more traditional approach to GAs became apparent. GAs’ find a few best solutions from many; by having a small n% reproduce, there are fewer increasingly specialised solutions each generation.

This products purpose was to grow many from few, this n% to reproduce had to be set a higher than prior research had suggested, raised to 30% during testing.

*APPENDIX:B07.01 – Selection Testing – declining populations*

The problem still persisted but at a reduced rate; further mechanics had to be implemented to fully correct the issue which will be explore in greater detail later.

Selection class was created, contained functions for parentSelection, vectors for selection tables and reproduction tables, alongside functions that filled these tables [*APPENDIX:C07.01-03*].

* Fill selection table with paired threshold score and variables of creatures.
* Sort this table of paired values by threshold score, highest to lowest.
* Take the top n%.
* Pass these selected paired values to toReproduceTable to be accessed by Crossover.
* Later extended to take creatures gene stacks.

Throughout the testing of this mechanic several issues were highlighted.

First identified was the importance of the starting population sizes. Initially set to 12 for each species; taking 30% to reproduce resulted in 4 creatures, two pairs of parents. This did not sustain population growth and was raised to 100 giving a large enough start to be sustainable.

The most important issue identified through testing of the selection mechanic was the approach taken in programming this product.

Initially all of the creatures were being stored in a single vector. When testing selection, it became apparent that creatures of completely different types were being paired and reproducing; a mess where different creatures were combined and some not at all.

At first the fix was to organise the creature vector by their creature IDs, then check in the selection to make sure paired creatures had the same ID. This proved to be complicated to implement and far from bulletproof, another approach was required.

A new mechanic, which entirely changed how the creatures were considered, was designed and implemented, a Species mechanic. This will be explored in detail later but put simply, instead of all creatures being stored together, each creature would be considered its own species, have its own species vector and be dealt with entirely separately from other species.

***Crossover Mechanic***

Creatures to reproduce have been selected, a process of combining their attributes together into new creature had to be developed. An approach inspired by earlier research was taken, “*A crossover point is selected within the genes for each pair of entities to be mated*”[[7]](#footnote-7) (Rawat et al, 2020).

The Selection mechanic leaves the program with a vector pair, threshold score and creatures’ attributes (later extended to gene stack) [*APPENDIX:C08.02*], sorted by threshold score; these are the parents to be combined.

Crossover process:

* Get parent creatures attributes.
* Into vector pair with a bool(false) for whether creature has been paired.
* Pair parents up, set bool(true).
* Get random int between 0 and length of creature attributes.
* Create a new creature.
* Assign parent 1 attributes up to random int.
* Assign parent 2 attributes from random int.
* Initialise creature attributes from these core attributes.
* Return new creature.
* Later extended to work with Gene Stacks.

At first was implemented with creatures’ core attributes (those used to initialise all creature attributes) but as the number of attributes increased, this approach became unwieldy. The core attributes were scattered across struct Creature and had to be hard-coded, then checked and altered every time a change elsewhere was made. Caused issues and little confidence in testing.

A further issue was highlighted when splitting child attributes and assigning parents attributes; a temporary vector was created to store these during this process to make it possible.

It became clear that this vector should not be temporary and purely in the crossover mechanic but rather an entirely new mechanic that made implementing crossover possible, the Gene Stack mechanic.

This will be explored in detail later but in short, a new vector of floats stored by every creature; each float a core variable that other variables could be initialised from. By keeping these core variables together, it made it far easier, controllable and elegant to implement the crossover mechanic.

***TESTING ‘Two Parent’ Approach***

All initial designs were now implemented but during this process the requirement for further mechanics had been identified. Still maintained the plan to run ‘Two Parent’ test as the follow up to the ‘Bacteria’ test to ensure selection and crossover working.

A shorter test was chosen as the infinite resource issue had been identified, just needed to ensure correct behaviour from selection and crossover. 10 cycles were chosen as allowed the observation of new generations to be reproduced, these were counted and the selection/crossover mechanics followed line by line to ensure correct implementation.

*APPENDIX:B09.01 – ‘Two Parent’ Stage Testing – imbalanced birth/death ratio.*

Testing gave confidence the selection and crossover mechanics were working as expected but it became apparent that the system itself was not working as required, populations were dying out after 5-10 cycles.

The birth/death ratios were imbalanced, causing steady and inevitable extinctions. The n% of the selection had been risen to 30%, raising any further risked removing any benefit of elitism selection method.

*APPENDIX:B09.03 – ‘Two Parent’ Stage Testing – birth/death ratio rebalance with life spans set to 2, yes, fitness tests applied.*

New mechanics were designed; Litter Sizes, the number of offspring creatures can have each cycle. Sitting alongside this was Life Spans, creatures could live longer than a single cycle and have the opportunity to reproduce more than once.

The above data table showed the impact of life span change to 2; the system became far more stable and sustainable, as can be seen when compared below.

*APPENDIX:B09.04 – ‘Two Parent’ Stage Testing – comparison of approaches.*

***Mid-Development Mechanics***

***Reflections & Next Stage Planning***

At this stage in development, initial mechanics had been implemented. During testing these mechanics had all, eventually, been refined and each individually proven to work as required. But as a whole, the system proved unable to achieve the desired functionality.

*Populations, not individuals.*

A watershed moment in the development of this product was the concept of *Species*, not just as a mechanic but as a concept of thinking about the product being developed. There had been too much consideration towards each individual creature, this was the wrong approach; it did not appreciate the processes involved in natural selection that this product was attempting to simulate. When it comes to evolution, the overall population is the driving force, not individuals.

The Species mechanic not only improved the system computationally and enabled easier implementation of the key Selection mechanic. It also reflected a thought shift in how the creatures and populations were to be considered as a whole within the program.

*Declining Populations*

A concern was the issue of declining populations; it was inevitable that a population would eventually perish due to a consistently larger death rate than birth rate.

During development a number of potential mechanics had been considered to improve the product, add nuance and ultimately, build a sustainable system for the species to evolve in.

* *Creature Life Spans.*
* *Offspring Litter Sizes.*
* *Creature Sizes.*

*Infinite Resources*

Even with the above implemented there was still a major issue with the product. There was potential for unlimited exponential growth that not only failed to simulate evolution, it would eventually crash a computer. This was considered a fatal flaw.

Therefore, the resources available must not be infinite, the *Environmental Capacity* mechanic was designed to prevent this. If the environment had a capacity weight, then the creatures required a *Creature Weight* to consider.

A further extension to this capacity was consequences, what happened once the capacity was reached. This led to the design of *Environmental Status* and *Creature Competition* mechanics.

***Species Mechanic***

The species mechanic has been touched upon at several points already. This was a change to the initial designs that not only improved the system, it made the final product achievable.

Rather than group all creatures together in a single vector; each time a creature survives in the seed stage, it’s used as the source for a species, a vector is created and named after the creatures ID. This species vector is then placed in a vector containing all species.

Struct SpeciesInfo was implemented, contained all necessary variables for each species [*APPENDIX:C09.01*]. Struct AllSpecies created, through which all of the species could be accessed [*APPENDIX:C09.02*]. Class Species created, contained functions for creating, updating and displaying all species information, this was later extended to include the same functionality for species gene stacks [*APPENDIX:C09.03*].

The key functionality of this class was to createNewSpecies() and addCreatureToSpecies(), whilst also updateAllSpecies() for AllSpecies. Later extended to include the creation and updating of species’ specific gene stacks [*APPENDIX:C09.04*].

This approach meant that each species could be dealt with in turn without worry of cross species interactions. It allowed ease of access to individual populations as well as combined populations. Most importantly was the change in approach to the entire product that this mechanic represented, the consideration of populations rather than individuals.

***Gene Stack Mechanic***

An alternative method to storing the core creatures’ variables was required; enabling a cleaner implementation of the essential Selection and Crossover mechanics. An alternate approach ensured the program worked and made the system clearer, as these core creature variables were stored together.

A new variable geneStack was included in the Creature object; vector of floats, each element containing one of the core creature variables which the other creature variables would be initialised from.

The gene stacks were implemented within the creatureCreation(), extendibility always a design consideration; and a new createCreatureFromGeneStack() function implemented [*APPENDIX:C10.01*].

*Gene Stacks Benefits*

There were many benefits to the gene stack method which positively impacted the other mechanics.

*Crossover* – made crossover a simpler process; the new gene stack would be the size of parents’ gene stack, split point between parent 1 and parent 2 obtained by a random number between 0 and the size of the gene stack. Assigning took parent 1 gene stack begin() to random number, then parent 2 from random number to end().

*Mutation* – do not need to mutate every variable for a new creature, just mutate these core gene stack variables. These mutations are reflected in all variables when created from these core variables.

*Species* – two gene stacks variables created for each species, seedGeneStack and currentGeneStack [*APPENDIX:C09.01*]. Seed taken from the original seed creature the species was generated from. The current gene stack was the combined and averaged values for all members of the species. Comparison between the two allowed observation of how a species had evolved.

Further extension would be a species divergence check, checking if there are members of the species that are so different they could be considered a new species. This was not implemented at the time as further research had to be taken into what constituted a new species.

***Environment Weight Capacity Mechanic***

A key issue with the product that was preventing it from achieving the desired functionality was the infinite resources that were available, leading to infinite population growth; a finite capacity for resources was required.

With extendibility being a constant consideration during the initial design, the environments were simple to extend and implement this functionality. The two main variables required were a capacity value and the population weight value.

The next concern was setting a capacity value; it was decided that an appropriate and simple approach would be to multiply the energy available variable by a multiplier variable, easy to access and tune [*APPENDIX:C01.02*].

To gain the combined weight of the total population mass first a value was created by iterating through all creatures and adding weights to total; this was deemed computationally expensive. As every creature in a species had the same weight, species’ weights worked out by multiplying their weight by size of species, then each species weight combined to get the total [*APPENDIX:C11.01*].

***Creature Weight Mechanic***

Now there was a capacity to the environment, the creatures required a weight to compare with that capacity. Extendible nature of the design of the creatures allowed this new mechanic to be implemented without major redesign. An additional variable creatureWeight was added to Creature and initialised to 0.1f.

To add further intelligence and justification to creatures’ weight, a mechanic was designed called Creature Size.

***Creature Size Mechanic***

The environment had a capacity and creatures had a weight but the standard cross-all-creatures value of 0.1f felt an arbitrary value; more intelligence to this weight was required.

A mechanic that assigns each creature a size was chosen as an ideal solution; inspired by the nature the program attempts to stimulate (larger creatures require more resources and weigh heavier on an environments), it gave intelligence to the weight of each creature. It could also be extended to impact on other mechanics such as life spans, litter size and competition. As well as being useful to mechanics in the distant design stage, such as a predator/prey system.

Enum CreatureSize created, first with SMALL, MEDIUM, LARGE, later extended to include VERY\_SMALL and VERY\_LARGE; alongside a CreatureSize variable within the Creature struct to contain each creatures’ size [*APPENDIX:C12.01*].

Sizes were assigned within the creatureCreation(), a random number between 0 and 4 was generated which would determine the size (0-VERY\_SMALL -> 4-VERY\_LARGE). A switch/case statement was used to assign these; as well as the creatures’ weight being dependent on size, it had already been determined that new mechanics like life span and litter size would also be determined by creature size[*APPENDIX:C02.02*].

Once creatures’ weight had been determined, a multiplier was applied to the base 0.1f:

* VERY\_SMALL - base: 0.1f - 80% reduction \* 0.2f = 0.02f weight.
* SMALL - base: 0.1f - 40% reduction \* 0.6f = 0.06f weight.
* MEDIUM - base: 0.1f - no change - remained at 0.1f weight.
* LARGE - base: 0.1f - 40% increase \* 1.4f = 0.14f weight.
* VERY\_LARGE - base: 0.1f - 80% increase \* 1.8f = 0.18f weight.

The extendibility of the Creature, as well as the creatureCreation(), meant a straight forward implementation into the existing product.

***Environmental Status Mechanic***

Environment had its capacity and creatures had their weights; next step was to extend this new functionality to have consequences when capacity reached and exceeded.

Environmental Status gave states which the environment could exist in, these states in turn would determine rules for the creatures in the environment. Following a similar approach to the creature size mechanic, enum EnvironmentalStatus created containing these states.

At first ENERGY\_AVAILABLE/ENERGY\_NOT\_AVAILABLE, though this was quickly replaced with naming that reflected the situation, ABUNDANCE/FAMINE.

If population weight under the capacity, then ABUNDANCE; FAMINE if population weight above capacity. Checked each cycle, if ABUNDANCE then no penalty to energy available, if FAMINE then a large penalty to energy available; killing off creatures until population weight returned to more sustainable levels.

After testing, the system worked at reducing the populations very effectively but on closer inspection it became apparent only very small creatures were surviving, this was not the desired result.

*APPENDIX:B11.01 – Environmental Status Testing – famines killing off all but very small creatures..*

Evolution does not like cliff-edge changes, these lead to mass-extinction as there isn’t the opportunity to evolve; gradient changes are required, enum was extended to ABUNDANCE, SUSTAINABLE, PRESSURED, CRITICAL, FAMINE allowing this gradient effect.

A period of tuning and testing took place, as initially the system was too ‘boom-and-bust’, by refining the tuning values it gradually began to provide a more sustainable output as can be seen in the graph below.

*APPENDIX:B11.02 – Environmental Status Testing – tuning of values, comparisons between set ups.*

A disproportionate death rate in larger creatures persisted, the system did not offer any positives to being larger. Taking inspiration from natural selection and evolution; what do creatures do when there are limited resources? They compete for them.

***Creature Competition Mechanic***

At this point it was evolutionally beneficial to be smaller, reasons needed to be larger. When resources are limited, there’s competition for those resources, this mechanic was an attempt to reflects that.

Two main impactors on competition, EnvironmentalStatus and the CreatureSize; first would determine effect on all creatures, second would determine effects dependent on the creature size.

EnvironmentalStatus would have the following impact [*APPENDIX:C13.01-02*]:

* FAMINE: 99% of creatures to compete.
* CRITICAL: 60%
* PRESSURED: 30%
* SUSTAINABLE: 0%
* ABUNDANCE: 0% (bonus abundant state)

Species creature size would determine the n% of species population considered to be ‘competing’ [*APPENDIX:C13.03*]:

* VERY\_SMALL: 95%
* SMALL: 90%
* MEDIUM: 80%
* LARGE: 70%
* VERY\_LARGE: 60%

These two percentages would give the overall % of a species population considered to be ‘competing’ resulting in a reduction of available energy.

|  |  |  |  |
| --- | --- | --- | --- |
| STATUS | SIZE | % TO COMPETE | TO COMPETE  (*from Pop of 1000*) |
| Pressured (30%) | Large (70%) | 21% | 210 |
| Pressured (30%) | Very Small (95%) | 28.5% | 285 |
| Critical (60%) | Medium (80%) | 48% | 480 |
| Famine (99%) | Very Large (60%) | 59.4% | 594 |
| Famine (99%) | Very Small (95%) | 94% | 940 |
| Sustainable (0%) | Small (90%) | 0% | 0 |

*Tab01.EnvironmentStatus & Creature Size = % of Pop. to Compete.*

Competition class created with functions setEnvironmentalStatus(), getEnvironmentalStatus() and getPopulationToCompete(), along with required environment, creature and multiplier variables; all accessible and adjustable [*APPENDIX:C13.02*].

Creatures considered to be ‘competing’ had full fitness test ran against them with the energy available reduced; required an extension to the fitness functions [*APPENDIX:C03.05*]. Whole process was placed before the regular fitness tests to avoid unnecessary duplication of work.

This mechanic was tested in stages, was necessary to ensure each part was bulletproof before moving on, as would have been difficult to locate issues retrospectively in all mechanics at once. First, environment status confirmed, then creature size, separately then together. To trigger the right circumstances the population weight was hard-set to particular values [*APPENDIX:C03.04*] and checked that it was triggering the expected behaviour.

Competition changed the Environment Status mechanic; rather than energy available being reduced for all, creatures deemed to be ‘competing’ had an energy available penalty applied individually [*APPENDIX:C03.05*]. Fixed issue with overcapacity environments decimating populations to the extent majority were unable to recover; a more sustainable approach.

This mechanic had the benefits of a positive reason to be larger; more expensive being big but creature has the best opportunity when competing for limited resources.

***Creature Litter Size Mechanic***

A major issue encountered was the birth/death ratio being imbalanced, causing extinctions. The % of population to be parents maxed at 30%, any higher ran the risk of removing evolutionary benefits of elitism selection.

It was determined that the creatures litter size would require increasing. With creature size implemented, it was the perfect vehicle to give reasoning to the litter size variable.

Creature struct extended to include a litter size variable, alongside extensions to creatureCreation() and the gene stack mechanic [*APPENDIX:C02.02*]. The switch statement which set a creatures’ weight extended to include litter size, taking a base value of 4.0f and then multiplying depending on size:

* VERY\_SMALL: increase 250% - 10.
* SMALL: increase 50% - 6.
* MEDIUM: initial 4.
* LARGE: reduced 50% - 2.
* VERY\_LARGE: hard-set - 1.

Base litterSize originally set to 3.0f but meant the difference between LARGE and VERY\_LARGE creatures to be 1.5f and 1.0f, meant there was a 50% chance of a single offspring, too small a gap between the two sizes.

Creature cannot have half an offspring, this variable remained a float as the remaining point value (i.e., 0.6f from 2.6f) was converted to a % and used as the chance that the creature will have an extra offspring. For example, 2.7f offspring would mean a definite 2 offspring with a 70% chance that that cycle the creature would have 3 offspring.

VERY\_LARGE creatures hard-set to 1 offspring per cycle as design was to interweave this with the life span mechanic; larger creatures live longer and smaller creatures live shorter life spans as a method of encouraging alternate survival methods.

With all creatures having uniform life spans, quickly caused flooding of smaller creatures in the environment. The competition mechanic gave advantages to larger creatures once environmental capacity was reached, it was clear a life span mechanic was required to achieve balance and sustainability.

***Creature Life Span Mechanic***

Smaller creatures were dominating environments, another benefit to being big was required to balance the system. A life span mechanic was settled upon, interweaved with the litter size mechanic. This mimicked the creature litter size implementation but flipped so advantages were for larger creatures.

Design approach of product meant Creature, creatureCreation() and gene stacks were easily extendable. Implementation of the life spans followed a similar methodology to litter size mechanic [*APPENDIX:C02.01*], variable lifeSpan initialised to 4 and then multipliers applied dependent on creature size:

* VERY\_SMALL: hard-set - 1.
* SMALL: reduced 50% - 2.
* MEDIUM: randomised +/- 1 or keep same, equal chances of each.
* LARGE: increased 50% - 6.
* VERY\_LARGE: increased 250% - 10.

Combining litter size and life span lead to massive strides in solving the issue of declining populations; with tuning it corrected the birth/death ratio of the creature.

If population explosions do take place, then the Environment Status and Creature Competition mechanics were in place to halt and reserve over-population.

***User Display***

As this product is not a finished program but rather the driving mechanic that either a ‘god’ genre game or automation tool would be built upon, there wasn’t a great demand for a visual display. But to ensure that the correct procedures were taking place, the data needed to be observed.

*Part A – interactive console and data display.*

As the output of this product is data, there needs to be a way for the user to observe and compare this data; this was implemented within the console window. Here a user is prompted to input to observe statistically information and influence the product.

Date included:

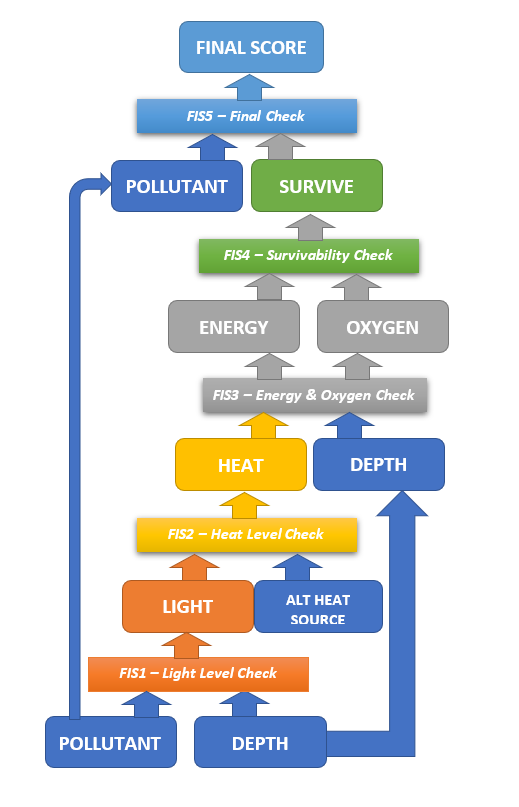
* Observe the evolution between Start-End gene stacks of species.
* Cycle-by-cycle population and species changes.
* Access the most successful species over all.
* Access the most successful species per turn.
* Access extinct species.

*Part B – primitive window displaying visual data.*

The second part was a planned window display using OpenGL. This would have displayed a stencil of a 2d rockpool and coloured the scene using a cloud shader, with different colours for different species, gaining in amount and intensity with increasing populations.

***Fuzzy Logic***

The aim was to implement the creation of the environmental variables using a type-1 Mamdani hierarchical fuzzy inference system that would be able to produce all required variables from a crisp depth value.



*Appendix:A1.01 – Design Diagram of Hierarchical FIS – dark blue crisp values.*

Considering extendibility and planned future mechanics, crisp values for a variable ‘Pollutant’ and ‘Alternate Heat’ were also included in the system, as these were planned user inputs. The system works as expected with just the single depth crisp value as it does with all crisp values included.

Five FIS’s were developed in a hierarchical structure:

1. Light Level Check
2. Heat Level Check
3. Energy & Oxygen Level Check
4. Survivability Check
5. Final Check.

The entire system was built and tested within MATLAB as that is purpose-built software for fuzzy implementation. All code is available here [APPENDIX:A2.01-05; APPENDIX:A3.01-05] and fully tested here [APPENDIX:A4.01-06].

Final step was to implement within this product; plan was to use software MATLAB-Coder. The hope was for this to be a straight-forward process; unfortunately, this was not the case and further hindered by the use of MATLAB-2019 version syntax that has become out-of-date since and incompatible with the latest version of MATLAB-Coder.

As a result, the fully designed, created and tested hierarchical FIS was not implemented within this product within the required timeframe. Though with a greater understanding of MATLAB-Coder this should be possible.

**CRITICAL EVALUATION *–* (*aim:1600 | is:2308)***

***Product Evaluation***

***How much does it do and how good is it?***

I am proud to say that the program does a great deal of what it was initially intended to; achieving this took a great detail of further design, development and implementation of more complex mechanics to sit alongside the initially designed mechanics.

The initial designs, implementing core GA principles of Fitness Tests, Selection, Crossover and Mutation, were developed and implemented as planned. Testing each part individually ensured that separately they worked. When testing as a whole, with the aim of producing eco-systems of evolving creatures, it quickly became apparent that this approach was not enough. Populations either exploded exponentially or became extinct.

Initially, I thought that there was an issue with these initial systems but after testing, they were behaving as expected. Issue was with the design; it did not provide the intelligence required to produce the desired functionality.

Implementing the more complicated, interlinked mechanics lead to this desired functionality; a varied sustainable evolving eco-system of creatures. There is intelligence and reasoning, it produces a varied and sustainable eco-system, the strength and variety of evolution can be tracked and compare; I would consider this product a success.

***Parts to be proud of?***

*Seed & First Populations.*

The design of the Seed and First Populations was implemented early in development and I’m still proud of it. As discussed, this product uses GAs in a different way, making many from few; but in the Seed stage GAs were implemented in a traditional sense, taking 15000 creatures and producing a handful of survivors for the First populations and origins for each species.

I felt that this was a powerful and effective way of not only producing creatures to start the product with, but to do so repeatedly with differing results.

*Species Mechanic*

I am proud of the species mechanic that was implemented. It’s relatively straight forward and simple to understand but made the implementation of Selection and Crossover possible.

More importantly is what the species mechanic represents in the thought process behind the entire product; the shifting of focus away from the individual and towards populations.

*Gene Stacks*

The design of the creature mechanic had the majority of its variables being initialised by a smaller group of core variables. The process of grouping these core variables into a single vector of floats called a geneStack was the next logical step and perhaps should have been a priority for the original set of designs and plans.

It’s the possibilities of the gene stacks, especially when combined with the species mechanic, that I am particularly proud of. In made possible observing evolution in each species by comparing an updated average species gene stack against its original seed gene stack.

To extend, this averaged species geneStack could be compared against individual creature gene stack, checking for potential divergence and new species.

*Crossover Mechanic*

The completed crossover mechanic was similar to many examples that I had previously researched. Because of the unusual approach taken with GAs, I often had to rethink and rework the implementation of the key components, especially selection.

But with crossover I could implement an approach that I had thoroughly researched and I feel the resulting code is simple, powerful and elegant.

*Interweaving of creature size, weight, life span and litter mechanics.*

Many of the processes of evolution that this product is attempting to simulate are not stand alone; often they are interlinked with many other considerations and a change to one can have wide ranging effects.

I really wanted to replicate this kind of interlinked nature to evolution and by being able to successfully combine the creature size into affecting the creatures’ weight, life span and litter size, offering a variety of pros and cons, was very satisfying.

***What potential issues exist within the product?***

*Beneficial to be smaller*

The issue still exists for the dominance of smaller creatures, this does reflection natural selection and evolution as there are far more smaller creatures in the world. Nature’s solution is the predator-prey reality we see around us. I feel that an implementation of a similar mechanic would go a long way to solving the explosions of smaller creatures.

*Lowering of energy demands*

Comparing end and start species gene stacks, observed an overriding trend for the evolution of lower energy demand. If this continued indefinitely, all species would eventually have the lowest possible energy demands; not a realistic reflection of the process the product is attempting to stimulate.

This is because the threshold score used to determine which creatures will be selected to reproduce is determined by the difference between the creatures’ energy demand and the environment energy available, therefore a lower energy demand means a higher threshold score and increased chance to pass on this evolutional trait.

The threshold score needs to take into account more than just excess remaining energy. With the implementation of a predator-prey mechanic new variables such as speed, strength and metabolism (all of which may require a higher energy demand) could be counted towards the threshold score.

Further thought, research and design required.

***What is not implemented?***

*OpenGL windowed Visual representation on the screen.*

Display only half finished, within the timeframe was unable to design and the implement a shader that would correctly display the information required in a pleasing manner.

*In product use of created Fuzzy Sets for environment creation.*

After all the work put into creating, testing, tuning and finalising a quite elegant hierarchical Type-1 Mamdani FIS for automated intelligent generation of environments. Designed to implement multiple environments dependant upon a crisp depth value and prepped for extension to take crisp Pollutant and Alternate Heat Source values.

It was a serious disappointment not to be able to correctly use MATLAB-Coder to convert it into C++ and use within this product. It was a misjudgement in approach to attempt to use MATLAB-Coder late in development without prior experience of using that software.

*Species divergence checks.*

I was unable within the timeframe to implement species divergence checks for sub or new species. Everything is in place, the idea of checking individual creature gene stacks against an averaged species gene stack all ready to go.

Main problem was that I could do decide on what constituted a new species; a whole new area of research and a more experienced opinion in biological divergence would have been greatly appreciated.

*User Changes – Pollution and Alternate Heat Source.*

User interaction with the environment, via the addition of ‘Pollution’ or ‘Alternate Heat Source’ has been setup with the required variables within the Environment objects and functions prepped. Final implementation of this was not included as it was deemed there was not enough time to correctly and fully test the product with this functionality.

Due to the changes that these additions would make to the environment and the importance of the environment as it is the problem the creature solutions are attempting to solve. It was felt it would be safer to not include until confident in the bulletproof nature of these mechanics.

***How would you extend this given time?***

*Multiple environments within one larger environment*

A top priority for future implementation would be environment variation. The current product has a uniform environment, this is not an accurate reflection of the problem it is attempting to simulate. A varied environment would also go further at encouraging evolution, divergence and variety.

The extension could be an array of environments, each initialised from crisp depth value, different areas with different conditions requiring different solutions from creatures.

This would be a more accurate reflection of the world and it would encourage further evolution. Different environments would offer the opportunities for different solutions. So long as anti-aliasing is employed where differing environments meet (evolution likes a gradual gradient, not cliff-edge changes) then evolution would be encouraged at a greater rate.

*Predator/Prey mechanic*

A top priority that would complete this product is a predator-prey mechanic.

This product is designed to simulate evolution; main forces for evolution are:

* Getting enough food.
* Reproducing.
* Staying alive.

This product does the first two but the staying alive part is only partially covered; it applies to herbivore creatures, getting their energy from the environment they are in.

By adding a predator-prey mechanic you complete the picture of evolution as you are implementing an evolutional arms race into the system; evolving to get food and evolving not to be food.

This would be a massive extension involving a lot of thought, design and preparation but would push this product to a far more complete and realistic stage.

It would also add the benefit of dealing with the bias towards smaller creatures, as creature size could be used to work out what can hunt what, a very small predator will be unable hunt a very large prey creature, another benefit to being large.

The overpopulation of smaller creatures could be eased as many of these would be prey. This would reduce the reliance on the Environmental Status and Creature Competition mechanics as these predators would not weigh on the environment as their resources do not depend directly on what the environment can provide. This would keep population weight sustainable.

A multitude of new mechanics required:

* Determine carnivores, omnivores and herbivores.
* Determine which creatures will hunt which other creatures, possible use of creature size.
* Determine whether a hunt is successful.
* Different hunting strategies; lone, pack, ambush etc.
* New variables required for movement speed, sense ability (sight, small, hearing), protection (hides, scales) and attack method (teeth, claws).

A complicated extension but an essential one for a more accurate simulation of evolution.

*Extension of Creature Variables*

If predator-prey mechanics were implemented a number of new variables and consideration for the creatures would be required.

Metabolism, fat stores, speed, strength, attack method (teeth, claws), defence (hinds), hunt strategies, defence strategies; the list could go on and on.

A benefit would be that each of these could be interlinked; a thick hide would be a good defence, increase cold temperature tolerance but reduce speed.

A great deal of thought and planning would need to be invested before attempting.

***Evaluation of Approach***

***How successful was the approach adopted?***

I feel that the approach that adopted at the start was appropriate at the time. With experience and lessons learnt by designing, developing and implementing this product, I would have taken a more informed approach. The key refinements being:

* If I thought it would take two hours, it took a day.
* If I thought it would take two days, it took a week.

A key element that was overlooked at the planning stage was the large number of new and extended mechanics required to achieved the desired functionality and nuance.

With hindsight, it’s clear that the initial system would only produce two outcomes, boom or bust, they succeeded massively or died out completely. These extra mechanics were essential for adding complexity and variation to the final product.

As a result of this approach, I was able to test each initial mechanic individually and fully; mid-development mechanics were often implemented as required and as a result, their testing wasn’t as thought through or thorough; often having to test several systems at once rather than individually first.

*Vectors vs Arrays & Object Pooling.*

A big decision was whether to use arrays or vectors for containing my populations of species and creatures.

Arrays have the benefit of being computationally less expensive that vectors, as they would be constantly used this was important to me. As their size must be set at compile time, a pooling approach could be taken, where a very large space could be reserved. Initially these were used and large pools created, a remanent of this approach remains in the Seed Population array and pool. The major issue was, what happens if a particularly successful species exceeds the reserved pool size of the vector.

This potentially product breaking issue was such a concern that it was felt that the safest approach would be to use vectors; these would allow changes in real-time and the use of .reserve() when sizes required were known brought down the computational expense.

*Too much work taking place on the Stack*

At the time of hand-in, there is too much work taking place on the stack and not on the heap. This is partially the result of the mid-development changes as they brought far more computational processes in the main.cpp. With time, these would be abstracted back to their own classes, tested and optimised to more ideal memory management.

***What did you learn by doing the project? And did I meet my academic objectives?***

My main academic objectives where to improved my C++ skills, explore an avenue of AI, get experience with tool building and understand the process and considerations of taking on a large project.

My C++ skills have certainly improved throughout the development of this project, especially using STL and the consideration of ensuring that code is extendible and easily understood. Returning to code many months after writing can be confusing, so ensuring that I had fully commented what I was doing saved a lot of time.

Another huge time saver was making sure that code was easily extendible; by doing this, especially with the Creature, gene stack and environments, made implementing the many extensions possible. If full redesigns were required with each change, I don’t think this product would have been possible within the timeframe.

Finally, I gained a great deal of experience in product management and development. This has taken almost 8 months to research, design and implement; despite this you never have as much time as you think. I gained more experience in taking on a large challenge, breaking it down into smaller and smaller manageable parts. Then working out and implementing each small task. Before combining them together into something large and quite complex; proves that with thought, anything is possible.

***How would I do it better if I was given the project again?***

First thing, I should have started implementation earlier. If I was to repeat this project, I would ensure that coded implementation started in November rather than January. You can do all the design, research and planning you want, you’re only ever going to really get to grips with problems that you didn’t consider once you start implementation, these problems become apparent quickly. Had I done that, there would have been more time to consider alternate approaches.

A watershed moment was the mind shift from Creatures to Species; this shifted the entire development and aided in focusing on what was important for the products functionality.

This reasoning can be extended to the gene stack; the creatures should have been implemented using this method from the start rather than mid-development.

***Evaluation of Tools Used***

***What language, libraries were used in development?***

C++ was a good choice, allowed me to upskill and was perfect for the more detailed memory management. OpenGL was a good choice as easy, accessible and I am experienced with it.

Bad choices were attempting to use MATLAB Coder and the FuzzyLite library. I am sure that both of these are fine products, I just let it too late to give myself time to familiarise myself enough to be able to confidently and safely use them.

***Final Thoughts***

What are my final thoughts….?

**REFERENCES**

Ali F. F., Nakao Z. & Chen Y. W., (2000) *Playing the Rock-Paper-Scissors game with a genetic algorithm*, Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512), La Jolla, CA, USA, 2000, pp. 741-745 vol.1.

Baba N., Nagasawa K., Kogawa K. & Handa H., (2009) *Environmental games made much more exciting by soft computing techniques*, 2009 ICCAS-SICE, Fukuoka, 2009, pp. 4977-4980.

Banzhaf W., Nordin P., Keller R. E. & Francone F. D., (1998) *Genetic Programming: An Introduction,* Morgan Kaufmann Publishers Inc., San Francisco, CA, 1998.

Barton N.H. & Coe J.B., (2009) *On the application of statistical physics to evolutionary biology,* Journal of Theoretical Biology, Volume 259, Issue 2, 2009, pp. 317-324.

BBC BiteSize (2020) Principles of Evolution by Natural Selection [Online] BBC Bitesize, <https://www.bbc.co.uk/bitesize/guides/zcqbdxs/revision/1> [Accessed 01/10/2020].

Buhusi, C. V., (1994) *Learning by simulating evolution in automatic fuzzy systems synthesis* Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference, Orlando, FL, 1994, pp. 1308-1313 vol.2.

Castillo, O., Lozano, A. & Melin, P., (2004) *Hierarchical genetic algorithms for fuzzy system optimization in intelligent control*, IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS '04., Banff, Alberta, Canada, 2004, pp. 292-297 Vol.1.

Chen Y., Chai L. H., Li C. Y. & Zhang Y. X., (2010) *Dynamic principle and simulation for evolution of biology*, 2010 Sixth International Conference on Natural Computation, Yantai, 2010, pp. 3928-3932.

Chen Y., Tang C. J., Li R., Zhu M. F., C. Li & Zuo J., (2010) *Reduced-GEP: Improving Gene Expression Programming by Gene Reduction*, 2010 Second International Conference on Intelligent Human-Machine Systems and Cybernetics, Nanjing, Jiangsu, 2010, pp. 176-179.

Cole N., Louis S. J. & Miles C., (2004) *Using a genetic algorithm to tune first-person shooter bots*, Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 139-145 Vol.1.

Darwin, C. (1859). *On the origin of species by means of natural selection, or, the preservation of favoured races in the struggle for life*. London: J. Murray, 1859.

Drageset O., Winands M. H. M., Gaina R. D. & Perez-Liebana D., (2019) *Optimising Level Generators for General Video Game AI*, 2019 IEEE Conference on Games (CoG), London, United Kingdom, 2019, pp. 1-8.

Ferreira C., (2006) *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, 2nd edn., The Netherlands: Springer, 2006.

Ferreira C., (2001) *Gene Expression Programming: a new adaptive algorithm for solving problems,* Complex Systems, 2001, 13 (2): pp. 87-129.

Fukuda, T., Hasegawa, Y. & Shimojima, K. (1995) *Structure organization of hierarchical fuzzy model using by genetic algorithm*, Proceedings of 1995 IEEE International Conference on Fuzzy Systems., Yokohama, Japan, 1995, pp. 295-300 vol.1

Gargolinski, S,. (2018) *Genetic Algorithms for Games Programming,* 2018.

Gavrilets S., (2004) *Fitness Landscapes and the Origin of Species,* Princeton University Press, Princeton, New Jersey, July 2004.

Gunturu, M., Shakarad, G. N., and Singh, (2017) S. *Fitness function to find game equilibria using genetic algorithms*, 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1531-1534.

Horii F., Hochin T. & Nomiya H., (2014) *Improvement of the Success Rate of Automatic Generation of Procedural Programs with Variable Initialization Using Genetic Programming*, 2014 IIAI 3rd International Conference on Advanced Applied Informatics, Kitakyushu, 2014, pp. 699-704.

Huo P., Shiu S. C. K., H. Wang & Niu B., (2009) *Application and Comparison of Particle Swarm Optimization and Genetic Algorithm in Strategy Defense Game*, 2009 Fifth International Conference on Natural Computation, Tianjin, 2009, pp. 387-392.

Joo, M. G. (2003) *A method of converting conventional fuzzy logic system to 2 layered hierarchical fuzzy system*, The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ '03., St Louis, MO, USA, 2003, pp. 1357-1362 vol.2

Jong-Hwan K & Hyun M, (1997) *Evolutionary programming techniques for constrained optimization problems*, in IEEE Transactions on Evolutionary Computation, vol. 1, no. 2, pp. 129-140, July 1997.

Kacprzyk, J. (1995) *Multistage control of a fuzzy system using a genetic algorithm*, Proceedings of 1995 IEEE International Conference on Fuzzy Systems., Yokohama, Japan, 1995, pp. 1083-1088 vol.3

Katsnelson I. M., Wolf I. Y. & Koonin E. V., (2018) *Towards physical principles of biological evolution*, Physica Scripta, Vol 93, Number 4, IOP Publishing Ltd, Feb 2018.

Kholimi A. S., Hamdani A. & Husniah L., (2018) *Automatic Game World Generation for Platformer Games Using Genetic Algorithm,* 2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Malang, Indonesia, 2018, pp. 495-498.

Kingsford, M. J., *Marine Ecosystem* [Online] Britannica <https://www.britannica.com/science/marine-ecosystem> [Accessed 02/10/20]

Koonin, E. V. (2012). *The logic of chance: The nature and origin of biological evolution*. Upper Saddle River, N.J: Pearson Education, 2012.

Koza J. R., (1999) *Darwinian invention and problem solving by means of genetic programming*, IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), Tokyo, Japan, 1999, pp. 604-609 vol.3.

Koza J. R., Bennett III F. H., Andre D. & Keane M. A., (1999) *Genetic Programming III Darwinian Invention and Problem Solving,* Morgan Kaufmann Publishers Inc., San Francisco, CA, 1999.

Kramer, M. D., & Zhang, D. (2000) *GAPS: a genetic programming system*, Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000, Taipei, Taiwan, 2000, pp. 614-619

Li T., Tang C., He T., Wu J. & Qin W., (2009) *Gene Expression Programming without Reduplicate Individu*als, 2009 Fifth International Conference on Natural Computation, Tianjin, 2009, pp. 249-253.

Lihong, R., Yongsheng, D. & Shihuang, S, (2000) *DNA genetic algorithms for design of fuzzy systems*, Ninth IEEE International Conference on Fuzzy Systems. FUZZ- IEEE 2000 (Cat. No.00CH37063), San Antonio, TX, USA, 2000, pp. 1005-1008 vol.2.

Martínez-Arellano G., Cant R. & Woods D., (2017) *Creating AI Characters for Fighting Games Using Genetic Programming*, in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 4, pp. 423-434, Dec. 2017.

Murata, T. & Ishibuchi, H., (1995) *Adjusting membership functions of fuzzy classification rules by genetic algorithms*, Proceedings of 1995 IEEE International Conference on Fuzzy Systems., Yokohama, Japan, 1995, pp. 1819-1824 vol.4.

Offutt J., Ammann P. & Liu L., (2006) *Mutation testing implements grammar-based testing*, Second Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006), Raleigh, NC, 2006, pp. 12-12.

Overholt, J. L., Cheok, K. C. & Smid, G. E. (1999) *Threshold fuzzy systems: a priority-based hierarchical control scheme*, Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251), San Diego, CA, 1999, pp. 2335-2339 vol.4.

Overholt, J. L. & Cheok, K. C. (2001) *Hierarchical systems control using threshold fuzzy systems*, 2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236), Tucson, AZ, USA, 2001, pp. 2257-2262 vol.4.

Rawat, S. S., Rawat, K. S., Rawat, V., & Nijhawan, R. (2020) *Neural Networks based Hand-crafted genetic learning approach to simulate Space Mario Game*, 2020 International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2020, pp. 1-5.

Ribeiro Filho, J. L., (1994) *The GAME system (Genetic Algorithms Manipulation Environment)*, IEE Colloquium on Applications of Genetic Algorithms, London, UK, 1994, pp. 2/1-2/4.

Salge C. & Miles C., (2008) *Using genetically optimized artificial intelligence to improve gameplaying fun for strategical games*, Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games, Los Angeles, California, ACM, 2008, pp. 7-14.

Sharma, S. K. & Tokhi, M. O., (2000) *Genetic evolution: a dynamic fuzzy approach*, Ninth IEEE International Conference on Fuzzy Systems. FUZZ- IEEE 2000 (Cat. No.00CH37063), San Antonio, TX, USA, 2000, pp. 748-752 vol.2

Stufflebeam, J., & Prasad, N. R. (1999) *Hierarchical fuzzy control*, FUZZ-IEEE'99. 1999 IEEE International Fuzzy Systems. Conference Proceedings (Cat. No.99CH36315), Seoul, South Korea, 1999, pp. 498-503 vol.1.

Thompson, M., & Jiggins, C. (2014) *Supergenes and their role in evolution*. Heredity 113, 1–8 (2014).

Wang, L. X., (1994) *A mathematical formulation of hierarchical systems using fuzzy logic systems*, Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference, Orlando, FL, 1994, pp. 183-188 vol.1.

Wang L., Mabu S., Ye F. & Hirasawa K., (2009) *Rule accumulation method with modified fitness function based on Genetic Network Programming*, 2009 ICCAS-SICE, Fukuoka, 2009, pp. 1000-1005.

Wang Z. & Ma H., (2008) *A Linear Genetic Programming with Reusable Gene*, 2008 International Symposium on Intelligent Information Technology Application Workshops, Shanghai, 2008, pp. 328-331.

Zadeh, L. A. (1965) ‘Fuzzy sets’, *Information and Control*, vol. 8, no. 3, pp. 338–353.

**APPENDICES**

# **TABLE OF CONTENTS**

***FUZZY LOGIC***

A01: Fuzzy Design Diagrams….…………….…….……….….………………………….……….pg45

A02: Hierarchical FIS 1-5 – CODE..….………….…………………………….………..……….pg46

A03: Fuzzy Inference Systems 1-5….………….…………………………….………..……….pg56

A04: Testing of FIS 1-5……………….….………….…………………………….………..……….pg64

***TESTING DATA***

B01: Population Test Tables…..………………….………………….………………….……….pg75

B02: Population Test Data…….………………….………………….……………………………pg76

B03: Mutation Test Tables…….………………….………………….…………………..………pg76

B04: Mutation Test Data…….………………….………………….………………….…….……pg79

B05: ‘Bacteria’ Stage Test Tables…….…….…….………………….………………..………pg81

B06: ‘Bacteria’ Stage Test Data…….………………….…….…….………………….….……pg84

B07: Selection Test Tables…….………………….………………….………………….….……pg87

B08: Selection Test Data…….………………….………………….………………….……….…pg87

B09: ‘Two Parent’ Stage Test Tables…….………………….………………..……….…….pg88

B10: ‘Two Parent’ Stage Test Data…….………………….………………….…………….…pg90

B11: Environmental Status Test Tables…….………………….……………….…………..pg91

B12: Environmental Status Test Data…….….………………….………………….…….…pg92

***CODE DATA***

C01: Environmental Creation ………………….………………….…………….……………..pg93

C02: Creature Creation ……….………………….………………….…………………………….pg94

C03: Fitness Tests……….……….………………….………………….………………….…..…….pg98

C04: Population Creation…….………………….………………….………………….…..…….pg102

C05:Mutation Mechanic.…….………………….………………….………………….…..…….pg102

C06: ‘Bacteria’ Stage Testing.………………….………………….………………….…..…….pg103

C07: Selection Mechanic.…….………………….………………….………………….…..…….pg104

C08: Crossover Mechanic…….………………….………………….………………….….….….pg106

C09: Species Mechanic….…….………………….………………….………………….….….….pg107

C10: Gene Stack Mechanic.….………………….………………….………………….….….….pg109

C11: Environmental Capacity Mechanic.….………………….…….….….……………….pg110

C12: Creature Size Mechanic.………………….………………….………………….…..…….pg110

C13: Creature Competition Mechanic.…….………………….……………….……..…….pg110

***DESIGNS***

D01: Initial System Designs……………….…….………………….……………….……..…….pg113

***DOCUMENTATION***

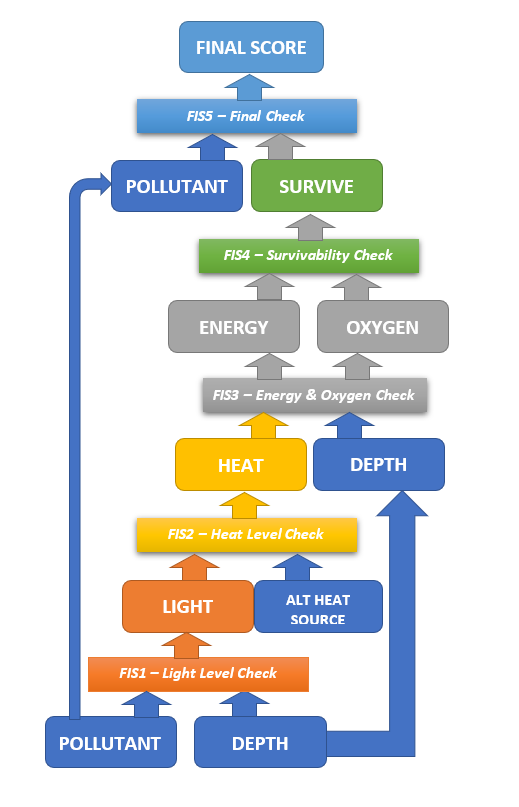
E01: Ethical Review…………..…..………………….………………….………………….……….pg116

E02: Global Checklist………..…..………………….………………….………………….……….pg118

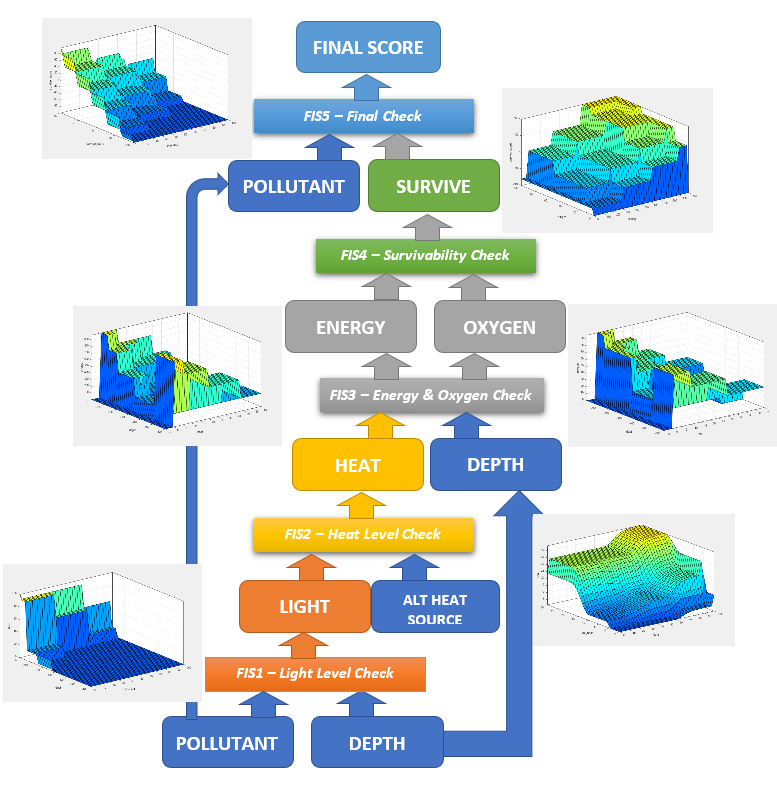
E03: Project Contract……….…..………………….………………….………………….………..pg120

***FUZZY LOGIC***

***A01: Fuzzy Design Diagrams***



*A1.01 – Fuzzy Design Diagram – flow of hierarchical mamdani type-1 fuzzy system for producing all required outputs from crisp values Depth & Pollutant*



*A1.02 – Fuzzy Design Diagram – flow of hierarchical mamdani type-1 fuzzy system for producing all required outputs from crisp values Depth & Pollutant. With surface charts.*

***A02: Hierarchical FIS - Code***

|  |
| --- |
| function FIS\_1  %first clear the screen.  %clc;  %var to hold excel file.  data1 = ('Data.xlsx');  %read data from xls file and store in testData.  inputData = xlsread(data1);  %{  FIS::1.1 - Light Check.  Input: Depth, Pollutate  Output: Light.  %}  %new Fuzzy Inference System called 'Light Level Check'.  fis1 = newfis('Light Level Check', 'DefuzzificationMethod', 'mom', 'AndMethod', 'min', ...  'ImplicationMethod', 'prod', 'AggregationMethod', 'max');  %declare 'Depth' INPUT variable.  fis1 = addvar(fis1, 'input', 'Depth (m)', [-600 0]);  %declare MFs for Depth input.  fis1 = addmf(fis1, 'input', 1, 'The Trenches', 'zmf', [-450 -400]);  fis1 = addmf(fis1, 'input', 1, 'The Abyss', 'gbellmf', [100 5 -350]);  fis1 = addmf(fis1, 'input', 1, 'Midnight Zone', 'gbellmf', [70 5 -200]);  fis1 = addmf(fis1, 'input', 1, 'Twilight Zone', 'gbellmf', [45 5 -95]);  fis1 = addmf(fis1, 'input', 1, 'Sunlight Zone', 'gbellmf', [30 5 -30]);  fis1 = addmf(fis1, 'input', 1, 'The Surface', 'smf', [-20 0]);  %declare 'Pollutant Present' INPUT variable.  fis1 = addvar(fis1, 'input', 'Pollutant Present (%)', [0 100]);  %declare MFs for Polluate Present input.  fis1 = addmf(fis1, 'input', 2, 'No Pollution', 'zmf', [0 3]);  fis1 = addmf(fis1, 'input', 2, 'Low Pollution', 'gbellmf', [17.5 3 15]);  fis1 = addmf(fis1, 'input', 2, 'Moderate Pollution', 'gaussmf', [15 50]);  fis1 = addmf(fis1, 'input', 2, 'High Pollution', 'gbellmf', [17.5 3 85]);  fis1 = addmf(fis1, 'input', 2, 'Complete Pollution', 'smf', [95 100]);  %declare the OUTPUT variable Light Level Check.  fis1 = addvar(fis1, 'output', 'Light Level (%)', [0 100]);  %declare MFs for Light Levels output variable.  fis1 = addmf(fis1, 'output', 1, 'No Light', 'trapmf', [0 0 0 1]);  fis1 = addmf(fis1, 'output', 1, 'Low Light', 'pimf', [0 0.5 30 55]);  fis1 = addmf(fis1, 'output', 1, 'Moderate Light', 'gaussmf', [15 55]);  fis1 = addmf(fis1, 'output', 1, 'High Light', 'pimf', [60 85 99 100]);  fis1 = addmf(fis1, 'output', 1, 'Complete Light', 'smf', [98 100]);  %RULES(18) In:DEPTH(5), In:POLLUTANT(6), Out:LIGHT(5), Weight=1, (AND) // 5 x 6 = 30 rules max  rule1 = [1 0 1 1 1]; %[In:The Trenches, In:ANY Pollution, Out:No Light, (1), AND] = covers all 5 rules for The Trenches  rule2 = [2 5 1 1 2]; %[In:The Abyss, In:Complete Pollution, Out:No Light, (1), OR] = by using OR, covers above two rules - this single rule covers 9 possible rules.  rule3 = [3 1 2 1 1]; %[In:Midnight Zone, In:No Pollution, Out:Low Light, (1), AND]  rule4 = [3 2 1 1 1]; %[In:Midnight Zone, In:Low Pollution, Out:No Light, (1), AND]  rule5 = [3 3 1 1 1]; %[In:Midnight Zone, In:Moderate Pollution, Out:No Light, (1), AND]  rule6 = [3 4 1 1 1]; %[In:Midnight Zone, In:High Pollution, Out:No Light, (1), AND]  rule7 = [4 1 2 1 1]; %[In:Twilight Zone, In:No Pollution, Out:Low Light, (1), AND]  rule8 = [4 2 2 1 1]; %[In:Twilight Zone, In:Low Pollution, Out:Low Light, (1), AND]  rule9 = [4 3 1 1 1]; %[In:Twilight Zone, In:Moderate Pollution, Out:No Light, (1), AND]  rule10 = [4 4 1 1 1]; %[In:Twilight Zone, In:High Pollution, Out:No Light, (1), AND]  rule11 = [5 1 4 1 1]; %[In:Sunlight Zone, In:No Pollution, Out:High Light, (1), AND]  rule12 = [5 2 4 1 1]; %[In:Sunlight Zone, In:Low Pollution, Out:High Light, (1), AND]  rule13 = [5 3 3 1 1]; %[In:Sunlight Zone, In:Moderate Pollution, Out:Moderate Light, (1), AND]  rule14 = [5 4 2 1 1]; %[In:Sunlight Zone, In:High Pollution, Out:Low Light, (1), AND]  rule15 = [6 1 5 1 1]; %[In:The Surface, In:No Pollution, Out:Complete Light, (1), AND]  rule16 = [6 2 4 1 1]; %[In:The Surface, In:Low Pollution, Out:High Light, (1), AND]  rule17 = [6 3 4 1 1]; %[In:The Surface, In:Moderate Pollution, Out:High Light, (1), AND]  rule18 = [6 4 3 1 1]; %[In:The Surface, In:High Pollution, Out:Moderate Light, (1), AND]  %rule list, placing the individual ARRAYS into a MATRIX  lightLevelRuleList = [rule1; rule2; rule3; rule4; rule5; rule6; rule7; rule8; rule9; rule10;  rule11; rule12; rule13; rule14; rule15; rule16; rule17; rule18 ];  %addrule passes the rulesList into out FIS (a).  fis1 = addRule(fis1, lightLevelRuleList);  %Display the rules of the FIS.  showrule(fis1)  %DATA OUTPUT - loop to run through the data, then output & write results.  for i=1:size(inputData, 1)  %var output to hold result of EVALFIS for the first 2 columns of INPUT data at row i in the excel document.  output = evalfis([inputData(i, 1), inputData(i, 2)], fis1);  %prints to console window, shows input values being fed into system.  fprintf('%d) In(1): %.2f, In(2) %.2f => Out: %.2f \n\n', i, inputData(i, 1), inputData(i, 2), output);  %write to result of FIS1 to InputData column E.  xlswrite('InputData.xlsx', output, 1, sprintf('E%d', i + 1));  end  end |

*A02.01 – Code – FIS1 – Light Levels Check*

|  |
| --- |
| function FIS\_2  %first clear the screen.  %clc;  %var to hold excel file.  data1 = ('Data.xlsx');  %read data from xls file and store in testData.  inputData = xlsread(data1);  %{  FIS::2.1 - Heat Check.  Input: Light(Output::FIS::1.1), Alternate Heat Source.  Output: Heat.  %}  %new Fuzzy Inference System called 'Heat Level Check'.  fis2 = newfis('Heat Level Check', 'DefuzzificationMethod', 'centroid','AndMethod', 'prod', ...  'ImplicationMethod', 'prod', 'AggregationMethod', 'max')  %declare the 'Light Level' INPUT variable.  fis2 = addvar(fis2, 'input', 'Light Level (%)', [0 100]);  %declare MFs for Light input.  fis2 = addmf(fis2, 'input', 1, 'No Light', 'trapmf', [0 0 0 1]);  fis2 = addmf(fis2, 'input', 1, 'Low Light', 'pimf', [0 0.5 30 55]);  fis2 = addmf(fis2, 'input', 1, 'Moderate Light', 'gaussmf', [15 55]);  fis2 = addmf(fis2, 'input', 1, 'High Light', 'pimf', [60 85 99 100]);  fis2 = addmf(fis2, 'input', 1, 'Complete Light', 'smf', [98 100]);  %declare 'Alternate Heat Source' INPUT variable.  fis2 = addvar(fis2, 'input', 'Alternate Heat Source (c)', [-5 30]);  %declare MFs for Alternate Heat Source input.  fis2 = addmf(fis2, 'input', 2, 'None', 'trapmf', [-1.0 -0.5 0.5 1.0]);  fis2 = addmf(fis2, 'input', 2, 'Cold Current', 'gbellmf', [4.5 12 -5]);  fis2 = addmf(fis2, 'input', 2, 'Warm Current', 'pimf', [0 1 10 20]);  fis2 = addmf(fis2, 'input', 2, 'Thermal Vents', 'smf', [10 20]);  %declare the OUTPUT variable Heat Level Check.  fis2 = addvar(fis2, 'output', 'Heat Level (c)', [-5 50]);  %declare MFs for Heat Levels output variable.  fis2 = addmf(fis2, 'output', 1, 'Freezing', 'trapmf', [-5 -5 0 0]);  fis2 = addmf(fis2, 'output', 1, 'Cold', 'trapmf', [0 0.5 2 3]);  fis2 = addmf(fis2, 'output', 1, 'Cool', 'pimf', [1 3 6 11]);  fis2 = addmf(fis2, 'output', 1, 'Temperate', 'gbellmf', [6.5 4 13]);  fis2 = addmf(fis2, 'output', 1, 'Warm', 'gbellmf', [7 4 24]);  fis2 = addmf(fis2, 'output', 1, 'Hot', 'gbellmf', [5 4 34]);  fis2 = addmf(fis2, 'output', 1, 'Very Hot', 'smf', [35 40]);  %RULES(20) In:LIGHT(5), In:ALTERNATE\_HEAT(4), Out:HEAT(7), Weight=1, (AND) // 5 x 4 = 20 rules max  rule1 = [1 1 2 1 1]; %[In:No Light, In:None, Out:Cold, (1), AND]  rule2 = [1 2 2 1 1]; %[In:No Light, In:Cold Current, Out:Cold, (1), AND]  rule3 = [1 3 3 1 1]; %[In:No Light, In:Warm Current, Out:Cool, (1), AND]  rule4 = [1 4 5 1 1]; %[In:No Light, In:Thermal Vents, Out:Warm, (1), AND]  rule5 = [2 1 2 1 1]; %[In:Low Light, In:None, Out:Cold, (1), AND]  rule6 = [2 2 2 1 1]; %[In:Low Light, In:Cold Current, Out:Cold, (1), AND]  rule7 = [2 3 3 1 1]; %[In:Low Light, In:Warm Current, Out:Cool, (1), AND]  rule8 = [2 4 6 1 1]; %[In:Low Light, In:Thermal Vents, Out:Hot, (1), AND]  rule9 = [3 1 3 1 1]; %[In:Moderate Light, In:None, Out:Cool, (1), AND]  rule10 = [3 2 2 1 1]; %[In:Moderate Light, In:Cold Current, Out:Cold, (1), AND]  rule11 = [3 3 4 1 1]; %[In:Moderate Light, In:Warm Current, Out:Temperate, (1), AND]  rule12 = [3 4 6 1 1]; %[In:Moderate Light, In:Thermal Vents, Out:Hot, (1), AND]  rule13 = [4 1 4 1 1]; %[In:High Light, In:None, Out:Temperate, (1), AND]  rule14 = [4 2 4 1 1]; %[In:High Light, In:Cold Current, Out:Temperate, (1), AND]  rule15 = [4 3 5 1 1]; %[In:High Light, In:Warm Current, Out:Warm, (1), AND]  rule16 = [4 4 7 1 1]; %[In:High Light, In:Thermal Vents, Out:Very Hot, (1), AND]  rule17 = [5 1 5 1 1]; %[In:Complete Light, In:None, Out:Warm, (1), AND]  rule18 = [5 2 4 1 1]; %[In:Complete Light, In:Cold Current, Out:Temperate, (1), AND]  rule19 = [5 3 6 1 1]; %[In:Complete Light, In:Warm Current, Out:Hot, (1), AND]  rule20 = [5 4 7 1 1]; %[In:Complete Light, In:Thermal Vents, Out:Very Hot, (1), AND]  %rule list, placing the individual ARRAYS into a MATRIX  heatLevelRuleList = [ rule1; rule2; rule3; rule4; rule5; rule6; rule7; rule8; rule9; rule10;  rule11; rule12; rule13; rule14; rule15; rule16; rule17; rule18; rule19; rule20 ];  %addrule passes the rulesList into out FIS (a).  fis2 = addRule(fis2, heatLevelRuleList);  %Display the rules of the FIS.  showrule(fis2)  %DATA OUTPUT - loop to run through the data, then output & write results.  for i=1:size(inputData, 1)  %var output to hold result of EVALFIS for the first 2 columns of INPUT data at row i in the excel document.  output = evalfis([inputData(i, 5), inputData(i, 3)], fis2);  %prints to console window, shows input values being fed into system.  fprintf('%d) In(1): %.2f, In(2) %.2f => Out: %.2f \n\n', i, inputData(i, 5), inputData(i, 3), output);  %write to result of FIS1 to InputData column E.  xlswrite('InputData.xlsx', output, 1, sprintf('F%d', i + 1));  end  end |

*A02.02 – Code – FIS2 – Heat Levels Check*

|  |
| --- |
| function FIS\_3  %first clear the screen.  %clc;  %var to hold excel file.  data1 = ('Data.xlsx');  %read data from xls file and store in testData.  inputData = xlsread(data1);  %{  FIS::3.1 - Energy Available & Oxygenation Checks.  Input: Heat(Output::FIS::2.1), Depth(Input::FIS::1.1)  Output: Energy, Oxygen.  %}  %new Fuzzy Inference System called 'Energy & Oxygen Check'.  fis3 = newfis('Energy & Oxygen Check', 'DefuzzificationMethod', 'mom', 'AndMethod', 'min', ...  'ImplicationMethod', 'prod', 'AggregationMethod', 'max')  %declare 'Depth' INPUT variable.  fis3 = addvar(fis3, 'input', 'Depth (m)', [-600 0]);  %declare MFs for Depth input.  fis3 = addmf(fis3, 'input', 1, 'The Trenches', 'zmf', [-450 -400]);  fis3 = addmf(fis3, 'input', 1, 'The Abyss', 'gbellmf', [100 5 -350]);  fis3 = addmf(fis3, 'input', 1, 'Midnight Zone', 'gbellmf', [70 5 -200]);  fis3 = addmf(fis3, 'input', 1, 'Twilight Zone', 'gbellmf', [45 5 -95]);  fis3 = addmf(fis3, 'input', 1, 'Sunlight Zone', 'gbellmf', [30 5 -30]);  fis3 = addmf(fis3, 'input', 1, 'The Surface', 'smf', [-20 0]);  %declare 'Heat Level' INPUT variable.  fis3 = addvar(fis3, 'input', 'Heat Level (c)', [-5 50]);  %declare MFs for Heat Levels input variable.  fis3 = addmf(fis3, 'input', 2, 'Freezing', 'trapmf', [-5 -5 0 0]);  fis3 = addmf(fis3, 'input', 2, 'Cold', 'trapmf', [0 0.5 4.5 6]);  fis3 = addmf(fis3, 'input', 2, 'Cool', 'pimf', [4 6 9 12]);  fis3 = addmf(fis3, 'input', 2, 'Temperate', 'gbellmf', [5 4 15]);  fis3 = addmf(fis3, 'input', 2, 'Warm', 'gbellmf', [6 4 24]);  fis3 = addmf(fis3, 'input', 2, 'Hot', 'gbellmf', [5 4 34]);  fis3 = addmf(fis3, 'input', 2, 'Very Hot', 'smf', [35 40]);  %declare the OUTPUT variable 'Energy Available'.  fis3 = addvar(fis3, 'output', 'Energy Available (J)', [0 1000]);  %declare MFs for Energy output variable.  fis3 = addmf(fis3, 'output', 1, 'Barren', 'zmf', [10 100]); %was 50/250 too much for barren  fis3 = addmf(fis3, 'output', 1, 'Sparse', 'gbellmf', [135 4 200]); %was too steep at 5, not enough cross over, back to 4. //was 200/2/225 shifted after change to barren, 2 to 4 to steeped curve into moderate as was leaking to far along the graph.  fis3 = addmf(fis3, 'output', 1, 'Moderate', 'gbellmf', [150 4 480]); %same as above // was 200/2/500 - needed 2-5 as needed to steepen curves, tails reaching too far.  fis3 = addmf(fis3, 'output', 1, 'Abundant', 'gbellmf', [145 4 760]); %same as above // was 200/2/775 - similar reasons for change as above.  fis3 = addmf(fis3, 'output', 1, 'Eden', 'smf', [850 950]); %same as above // was 750/950 - 750 reached too far back to be 'eden'  %declare the OUTPUT variable 'Oxygenation Rate'.  fis3 = addvar(fis3, 'output', 'Oxygenation Rate (%)', [0 100]);  %declare MFs for Oxygen output variable.  fis3 = addmf(fis3, 'output', 2, 'Death Zone', 'zmf', [0.25 1]); %reduced from 2/8  fis3 = addmf(fis3, 'output', 2, 'Very Low', 'gbellmf', [12 2.5 10]);  fis3 = addmf(fis3, 'output', 2, 'Low', 'gbellmf', [14 3 30]);  fis3 = addmf(fis3, 'output', 2, 'Moderate', 'gbellmf', [14 3 55]);  fis3 = addmf(fis3, 'output', 2, 'High', 'gbellmf', [14 3 78]);  fis3 = addmf(fis3, 'output', 2, 'Very High', 'smf', [83 92]);  %RULES(37) In:DEPTH(6), In:HEAT(7), Out:ENERGY(5), Out:OXYGEN(6), Weight=1, (AND) // 6 x 7 = 42.  rule1 = [0 1 1 1 1 1]; %[In:Any, In:Freezing, Out:Barren, Out:Death Zone, (1), AND] %takes care of freezing, so 1 rule for 7.  rule2 = [1 2 5 5 1 1]; %[In:The Trenches, In:Cold, Out:Eden, Out:High, (1), AND]  rule3 = [1 3 4 5 1 1]; %[In:The Trenches, In:Cool, Out:Abundant, Out:High, (1), AND]  rule4 = [1 4 4 3 1 1]; %[In:The Trenches, In:Temperate, Out:Abundant, Out:Moderate, (1), AND]  rule5 = [1 5 3 3 1 1]; %[In:The Trenches, In:Warm, Out:Moderate, Out:Moderate, (1), AND]  rule6 = [1 6 3 2 1 1]; %[In:The Trenches, In:Hot, Out:Moderate, Out:Low, (1), AND]  rule7 = [1 7 2 2 1 1]; %[In:The Trenches, In:Very Hot, Out:Sparse, Out:Low, (1), AND]  rule8 = [2 2 2 3 1 1]; %[In:The Abyss, In:Cold, Out:Sparse, Out:Low, (1), AND]  rule9 = [2 3 2 3 1 1]; %[In:The Abyss, In:Cool, Out:Sparse, Out:Low, (1), AND]  rule10 = [2 4 1 2 1 1]; %[In:The Abyss, In:Temperate, Out:Barren, Out:Very Low, (1), AND]  rule11 = [2 5 1 2 1 1]; %[In:The Abyss, In:Warm, Out:Barren, Out:Very Low, (1), AND]  rule12 = [2 6 1 2 1 1]; %[In:The Abyss, In:Hot, Out:Barren, Out:Very Low, (1), AND]  rule13 = [2 7 1 1 1 1]; %[In:The Abyss, In:Very Hot, Out:Barren, Out:Death Zone, (1), AND]  rule14 = [3 2 3 5 1 1]; %[In:Midnight Zone, In:Cold, Out:Moderate, Out:High, (1), AND]  rule15 = [3 3 3 4 1 1]; %[In:Midnight Zone, In:Cool, Out:Moderate, Out:Moderate, (1), AND]  rule16 = [3 4 2 4 1 1]; %[In:Midnight Zone, In:Temperate, Out:Sparse, Out:Moderate, (1), AND]  rule17 = [3 5 2 3 1 1]; %[In:Midnight Zone, In:Warm, Out:Sparse, Out:Low, (1), AND]  rule18 = [3 6 1 2 1 1]; %[In:Midnight Zone, In:Hot, Out:Barren, Out:Very Low, (1), AND]  rule19 = [3 7 1 2 1 1]; %[In:Midnight Zone, In:Very Hot, Out:Barren, Out: Very Low, (1), AND]  rule20 = [4 2 4 5 1 1]; %[In:Twilight Zone, In:Cold, Out:Adundant, Out:High, (1), AND]  rule21 = [4 3 4 5 1 1]; %[In:Twilight Zone, In:Cool, Out:Adundant, Out:High, (1), AND]  rule22 = [4 4 3 4 1 1]; %[In:Twilight Zone, In:Temperate, Out:Moderate, Out:Moderate, (1), AND]  rule23 = [4 5 3 4 1 1]; %[In:Twilight Zone, In:Warm, Out:Moderate, Out:Moderate, (1), AND]  rule24 = [4 6 2 3 1 1]; %[In:Twilight Zone, In:Hot, Out:Sparse, Out:Low, (1), AND]  rule25 = [4 7 1 3 1 1]; %[In:Twilight Zone, In:Very Hot, Out:Barren, Out:Low, (1), AND]  rule26 = [5 2 5 6 1 1]; %[In:Sunlight Zone, In:Cold, Out:Eden, Out:Very High, (1), AND]  rule27 = [5 3 4 5 1 1]; %[In:Sunlight Zone, In:Cool, Out:Abundant, Out:High, (1), AND]  rule28 = [5 4 4 5 1 1]; %[In:Sunlight Zone, In:Temperate, Out:Abundant, Out:High, (1), AND]  rule29 = [5 5 3 4 1 1]; %[In:Sunlight Zone, In:Warm, Out:Moderate, Out:Moderate, (1), AND]  rule30 = [5 6 3 3 1 1]; %[In:Sunlight Zone, In:Hot, Out:Moderate, Out:Low, (1), AND]  rule31 = [5 7 2 3 1 1]; %[In:Sunlight Zone, In:Very Hot, Out:Sparse, Out:Low, (1), AND]  rule32 = [6 2 5 6 1 1]; %[In:The Surface, In:Cold, Out:Eden, Out:Very High, (1), AND]  rule33 = [6 3 5 6 1 1]; %[In:The Surface, In:Cool, Out:Eden, Out:Very High, (1), AND]  rule34 = [6 4 4 5 1 1]; %[In:The Surface, In:Temperate, Out:Abundant, Out:High, (1), AND]  rule35 = [6 5 4 4 1 1]; %[In:The Surface, In:Warm, Out:Abundant, Out:Moderate, (1), AND]  rule36 = [6 6 3 3 1 1]; %[In:The Surface, In:Hot, Out:Moderate, Out:Low, (1), AND]  rule37 = [6 7 2 3 1 1]; %[In:The Surface, In:Very Hot, Out:Sparse, Out:Low, (1), AND  %rule list, placing the individual ARRAYS into a MATRIX  eneryOxygenLevelRuleList = [ rule1; rule2; rule3; rule4; rule5; rule6; rule7; rule8; rule9; rule10;  rule11; rule12; rule13; rule14; rule15; rule16; rule17; rule18; rule19; rule20; rule21; rule22;  rule23; rule24; rule25; rule26; rule27; rule28; rule29; rule30; rule31; rule32; rule33; rule34;  rule35; rule36; rule37 ];  %addrule passes the rulesList into FIS.  fis3 = addRule(fis3, eneryOxygenLevelRuleList);  %Display the rule  showrule(fis3)  %DATA OUTPUT - loop to run through the data, then output & write results.  for i=1:size(inputData, 1)  %var output to hold result of EVALFIS for the first 2 columns of INPUT data at row i in the excel document.  output = evalfis([inputData(i, 1), inputData(i, 6)], fis3);  %prints to console window, shows input values being fed into system.  fprintf('%d) In(1): %.2f, In(2) %.2f => Out: %.2f \n\n', i, inputData(i, 1), inputData(i, 6), output);  %write to result of FIS1 to InputData column E.  xlswrite('InputData.xlsx', output, 1, sprintf('G%d', i + 1));  end  end |

*A02.03 – Code – FIS3 – Energy & Oxygen Levels Check*

|  |
| --- |
| function FIS\_4  %first clear the screen.  %clc;  %var to hold excel file.  data1 = ('Data.xlsx');  %read data from xls file and store in testData.  inputData = xlsread(data1);  %{  FIS::4.1 - Survivability Check.  Input: Polluntant(Input::FIS::1.1), Energy Available(Output::FIS::3.1), Oxygen(Output::FIS::3.2)  Output: FINAL::Survivability.  %}  %new Fuzzy Inference System called 'Survivability Check'.  fis4 = newfis('Survivability Check', 'DefuzzificationMethod', 'lom', 'AndMethod', 'prod', ...  'ImplicationMethod', 'prod', 'AggregationMethod', 'max');  %declare the INPUT variable 'Energy Available'.  fis4 = addvar(fis4, 'input', 'Energy Available (J)', [0 1000]);  %declare MFs for Energy output variable.  fis4 = addmf(fis4, 'input', 1, 'Barren', 'zmf', [10 100]);  fis4 = addmf(fis4, 'input', 1, 'Sparse', 'gbellmf', [135 4 200]);  fis4 = addmf(fis4, 'input', 1, 'Moderate', 'gbellmf', [150 4 480]);  fis4 = addmf(fis4, 'input', 1, 'Abundant', 'gbellmf', [145 4 760]);  fis4 = addmf(fis4, 'input', 1, 'Eden', 'smf', [850 950]);  %declare the OUTPUT variable 'Oxygenation Rate'.  fis4 = addvar(fis4, 'input', 'Oxygenation Rate (%)', [0 100]);  %declare MFs for Oxygen output variable.  fis4 = addmf(fis4, 'input', 2, 'Death Zone', 'zmf', [0.25 1]);  fis4 = addmf(fis4, 'input', 2, 'Very Low', 'gbellmf', [12 2.5 10]);  fis4 = addmf(fis4, 'input', 2, 'Low', 'gbellmf', [14 3 30]);  fis4 = addmf(fis4, 'input', 2, 'Moderate', 'gbellmf', [14 3 55]);  fis4 = addmf(fis4, 'input', 2, 'High', 'gbellmf', [14 3 78]);  fis4 = addmf(fis4, 'input', 2, 'Very High', 'smf', [83 92]);  %declare the OUTPUT variable 'Survivability'.  fis4 = addvar(fis4, 'output', 'Survive Chance', [-100 100]);  %declare MFs for output variable.  fis4 = addmf(fis4, 'output', 1, 'Impossible', 'zmf', [-98 -95]);  fis4 = addmf(fis4, 'output', 1, 'Very Low', 'gaussmf', [25 -80]);  fis4 = addmf(fis4, 'output', 1, 'Low', 'gaussmf', [25 -40]);  fis4 = addmf(fis4, 'output', 1, 'Possible', 'gaussmf', [25 0 ]);  fis4 = addmf(fis4, 'output', 1, 'High', 'gaussmf', [25 40]);  fis4 = addmf(fis4, 'output', 1, 'Very High', 'gaussmf', [25 80]);  fis4 = addmf(fis4, 'output', 1, 'Guaranteed', 'smf', [95 98]);  %RULES(26)rules76-101: In:ENERGY(5), In:Oxygen(6), Out:Survivability(7), Weight=1,(AND) // 5 x 6 = 30.  rule1 = [0 1 1 1 1]; %[In:Any, In:Death Zone, Out:Impossible, (1), AND]  rule2 = [1 2 2 1 1]; %[In:Barren, In:Very Low, Out:Very Low, (1), AND]  rule3 = [1 3 2 1 1]; %[In:Barren, In:Low, Out:Very Low, (1), AND]  rule4 = [1 4 2 1 1]; %[In:Barren, In:Moderate, Out:Very Low, (1), AND]  rule5 = [1 5 2 1 1]; %[In:Barren, In:High, Out:Very Low, (1), AND]  rule6 = [1 6 2 1 1]; %[In:Barren, In:Very High, Out:Very Low, (1), AND]  rule7 = [2 2 2 1 1]; %[In:Sparse, In:Very Low, Out:Very Low, (1), AND]  rule8 = [2 3 3 1 1]; %[In:Sparse, In:Low, Out:Low, (1), AND]  rule9 = [2 4 3 1 1]; %[In:Sparse, In:Moderate, Out:Low, (1), AND]  rule10 = [2 5 4 1 1]; %[In:Sparse, In:High, Out:Possible, (1), AND]  rule11 = [2 6 4 1 1]; %[In:Sparse, In:Very High, Out:Possible, (1), AND]  rule12 = [3 2 3 1 1]; %[In:Moderate, In:Very Low, Out:Low, (1), AND]  rule13 = [3 3 4 1 1]; %[In:Moderate, In:Low, Out:Possible, (1), AND]  rule14 = [3 4 4 1 1]; %[In:Moderate, In:Moderate, Out:Possible, (1), AND]  rule15 = [3 5 5 1 1]; %[In:Moderate, In:High, Out:High, (1), AND]  rule16 = [3 6 5 1 1]; %[In:Moderate, In:Very High, Out:High, (1), AND]  rule17 = [4 2 4 1 1]; %[In:Abundant, In:Very Low, Out:Possible, (1), AND]  rule18 = [4 3 5 1 1]; %[In:Abundant, In:Low, Out:High, (1), AND]  rule19 = [4 4 6 1 1]; %[In:Abundant, In:Moderate, Out:Very High, (1), AND]  rule20 = [4 5 6 1 1]; %[In:Abundant, In:High, Out:Very High, (1), AND]  rule21 = [4 6 7 1 1]; %[In:Abundant, In:Very High, Out:Guaranteed, (1), AND]  rule22 = [5 2 5 1 1]; %[In:Eden, In:Very Low, Out:High, (1), AND]  rule23 = [5 3 6 1 1]; %[In:Eden, In:Low, Out:Very High, (1), AND]  rule24 = [5 4 6 1 1]; %[In:Eden, In:Moderate, Out:Very High, (1), AND]  rule25 = [5 5 7 1 1]; %[In:Eden, In:High, Out:Guaranteed, (1), AND]  rule26 = [5 6 7 1 1]; %[In:Eden, In:Very High, Out:Guaranteed, (1), AND]  %rule list, placing the individual ARRAYS into a MATRIX  firstSurviveCheckRuleList = [ rule1; rule2; rule3; rule4; rule5; rule6; rule7; rule8; rule9; rule10;  rule11; rule12; rule13; rule14; rule15; rule16; rule17; rule18; rule19; rule20; rule21; rule22;  rule23; rule24; rule25; rule26 ];  %addrule passes the rulesList into out FIS (a).  fis4 = addRule(fis4, firstSurviveCheckRuleList);  %Display the rules of the FIS.  showrule(fis4)  %DATA OUTPUT - loop to run through the data, then output & write results.  for i=1:size(inputData, 1)  %var output to hold result of EVALFIS for the first 2 columns of INPUT data at row i in the excel document.  output = evalfis([inputData(i, 7), inputData(i, 8)], fis4);  %prints to console window, shows input values being fed into system.  fprintf('%d) In(1): %.2f, In(2) %.2f => Out: %.2f \n\n', i, inputData(i, 7), inputData(i, 8), output);  %write to result of FIS1 to InputData column E.  xlswrite('InputData.xlsx', output, 1, sprintf('I%d', i + 1));  end  end |

*A02.04 – Code – FIS4 – Survivability Check*

|  |
| --- |
| function FIS\_5  %first clear the screen.  %clc;  %var to hold excel file.  data1 = ('Data.xlsx');  %read data from xls file and store in testData.  inputData = xlsread(data1);  %{  FIS::5.1 - Final Check.  Input: Polluntant(Input::FIS::1.1), Survivability(Output::FIS::4.1)  Output: FINAL::Final Score.  %}  %new Fuzzy Inference System called 'Final Check'.  fis5 = newfis('Final Check', 'DefuzzificationMethod', 'mom', 'AndMethod', 'min', ...  'ImplicationMethod', 'min', 'AggregationMethod', 'max')  %declare 'Pollutant Present' INPUT variable.  fis5 = addvar(fis5, 'input', 'Pollutant Present (%)', [0 100]);  %declare MFs for Polluate Present input.  fis5 = addmf(fis5, 'input', 1, 'No Pollution', 'zmf', [0 3]);  fis5 = addmf(fis5, 'input', 1, 'Low Pollution', 'gbellmf', [17.5 3 15]);  fis5 = addmf(fis5, 'input', 1, 'Moderate Pollution', 'gaussmf', [15 50]);  fis5 = addmf(fis5, 'input', 1, 'High Pollution', 'gbellmf', [17.5 3 85]);  fis5 = addmf(fis5, 'input', 1, 'Complete Pollution', 'smf', [95 100]);  %declare the INPUT variable 'Survivability'.  fis5 = addvar(fis5, 'input', 'Survive Chance', [-100 100]);  %declare MFs for input variable.  fis5 = addmf(fis5, 'input', 2, 'Impossible', 'zmf', [-98 -95]);  fis5 = addmf(fis5, 'input', 2, 'Very Low', 'gaussmf', [25 -80]);  fis5 = addmf(fis5, 'input', 2, 'Low', 'gaussmf', [25 -40]);  fis5 = addmf(fis5, 'input', 2, 'Possible', 'gaussmf', [25 0 ]);  fis5 = addmf(fis5, 'input', 2, 'High', 'gaussmf', [25 40]);  fis5 = addmf(fis5, 'input', 2, 'Very High', 'gaussmf', [25 80]);  fis5 = addmf(fis5, 'input', 2, 'Guaranteed', 'smf', [95 98]);  %declare the OUTPUT variable 'Final Score'.  fis5 = addvar(fis5, 'output', 'Final Score', [-100 100]);  %declare MFs for output variable.  fis5 = addmf(fis5, 'output', 1, 'Impossible', 'zmf', [-98 -95]);  fis5 = addmf(fis5, 'output', 1, 'Very Low', 'gaussmf', [25 -80]);  fis5 = addmf(fis5, 'output', 1, 'Low', 'gaussmf', [25 -40]);  fis5 = addmf(fis5, 'output', 1, 'Possible', 'gaussmf', [25 0 ]);  fis5 = addmf(fis5, 'output', 1, 'High', 'gaussmf', [25 40]);  fis5 = addmf(fis5, 'output', 1, 'Very High', 'gaussmf', [25 80]);  fis5 = addmf(fis5, 'output', 1, 'Guaranteed', 'smf', [95 98]);  %RULES(25) In:Pollution(5), In:Survivability(6), Out:Final Score(7), Weight=1,(AND) // 5 x 6 = 30.  rule1 = [5 1 1 1 2]; %[In:Complete Pollution, In:Impossible, Out:Impossible, (1), OR]  rule2 = [1 2 2 1 1]; %[In:No Pollution, In:Very Low, Out:Very Low, (1), AND]  rule3 = [1 3 3 1 1]; %[In:No Pollution, In:Low, Out:Low, (1), AND]  rule4 = [1 4 4 1 1]; %[In:No Pollution, In:Possible, Out:Possible, (1), AND]  rule5 = [1 5 5 1 1]; %[In:No Pollution, In:High, Out:High, (1), AND]  rule6 = [1 6 6 1 1]; %[In:No Pollution, In:Very High, Out:Very High, (1), AND]  rule7 = [1 7 7 1 1]; %[In:No Pollution, In:Guaranteed, Out:Guaranteed, (1), AND]  rule8 = [2 2 1 1 1]; %[In:Low Pollution, In:Very Low, Out:Impossible, (1), AND]  rule9 = [2 3 2 1 1]; %[In:Low Pollution, In:Low, Out:Very Low, (1), AND]  rule10 = [2 4 3 1 1]; %[In:Low Pollution, In:Possible, Out:Low, (1), AND]  rule11 = [2 5 4 1 1]; %[In:Low Pollution, In:High, Out:Possible, (1), AND]  rule12 = [2 6 5 1 1]; %[In:Low Pollution, In:Very High, Out:High, (1), AND]  rule13 = [2 7 6 1 1]; %[In:Low Pollution, In:Guaranteed, Out:Very High, (1), AND]  rule14 = [3 2 1 1 1]; %[In:Moderate Pollution, In:Very Low, Out:Impossible, (1), AND]  rule15 = [3 3 1 1 1]; %[In:Moderate Pollution, In:Low, Out:Impossible, (1), AND]  rule16 = [3 4 2 1 1]; %[In:Moderate Pollution, In:Possible, Out:Very Low, (1), AND]  rule17 = [3 5 3 1 1]; %[In:Moderate Pollution, In:High, Out:Low, (1), AND]  rule18 = [3 6 4 1 1]; %[In:Moderate Pollution, In:Very High, Out:Possible, (1), AND]  rule19 = [3 7 5 1 1]; %[In:Moderate Pollution, In:Guaranteed, Out:High, (1), AND]  rule20 = [4 2 1 1 1]; %[In:High Pollution, In:Very Low, Out:Impossible, (1), AND]  rule21 = [4 3 1 1 1]; %[In:High Pollution, In:Low, Out:Impossible, (1), AND]  rule22 = [4 4 1 1 1]; %[In:High Pollution, In:Possible, Out:Impossible, (1), AND]  rule23 = [4 5 2 1 1]; %[In:High Pollution, In:High, Out:Very Low, (1), AND]  rule24 = [4 6 3 1 1]; %[In:High Pollution, In:Very High, Out:Low, (1), AND]  rule25 = [4 7 4 1 1]; %[In:High Pollution, In:Guaranteed, Out:Possible, (1), AND]  %rule list, placing the individual ARRAYS into a MATRIX  firstSurviveCheckRuleList = [ rule1; rule2; rule3; rule4; rule5; rule6; rule7; rule8; rule9; rule10;  rule11; rule12; rule13; rule14; rule15; rule16; rule17; rule18; rule19; rule20; rule21; rule22;  rule23; rule24; rule25 ];  %addrule passes the rulesList into out FIS (a).  fis5 = addRule(fis5, firstSurviveCheckRuleList);  %Display the rules of the FIS.  showrule(fis5)  %DATA OUTPUT - loop to run through the data, then output & write results.  for i=1:size(inputData, 1)  %var output to hold result of EVALFIS for the first 2 columns of INPUT data at row i in the excel document.  output = evalfis([inputData(i, 2), inputData(i, 9)], fis5);  %prints to console window, shows input values being fed into system.  fprintf('%d) In(1): %.2f, In(2) %.2f => Out: %.2f \n\n', i, inputData(i, 2), inputData(i, 9), output);  %write to result of FIS1 to InputData column E.  xlswrite('InputData.xlsx', output, 1, sprintf('J%d', i + 1));  end  end |

*A02.05 – Code – FIS5 – Final Check*

***A03: Fuzzy Inference Systems***

|  |
| --- |
| [System]  Name='FIS1\_Testing'  Type='mamdani'  Version=2.0  NumInputs=2  NumOutputs=1  NumRules=18  AndMethod='min'  OrMethod='max'  ImpMethod='prod'  AggMethod='max'  DefuzzMethod='mom'  [Input1]  Name='depth'  Range=[-600 0]  NumMFs=6  MF1='trenches':'zmf',[-450 -400]  MF2='abyss':'gbellmf',[100 5 -350]  MF3='midnight':'gbellmf',[70 5 -200]  MF4='twilight':'gbellmf',[45 5 -95]  MF5='sunlight':'gbellmf',[30 5 -30]  MF6='surface':'smf',[-20 0]  [Input2]  Name='popullant'  Range=[0 100]  NumMFs=5  MF1='no\_pollution':'zmf',[0 3]  MF2='low\_pollution':'gbellmf',[17.5 3 15]  MF3='moderate\_pollution':'gaussmf',[15 50]  MF4='high\_pollution':'gbellmf',[17.5 3 85]  MF5='complete\_pollution':'smf',[95 100]  [Output1]  Name='light'  Range=[0 100]  NumMFs=5  MF1='no\_light':'trapmf',[0 0 0 1]  MF2='low\_light':'pimf',[0 0.5 30 55]  MF3='moderate\_light':'gaussmf',[15 55]  MF4='high\_light':'pimf',[60 85 99 100]  MF5='complete\_light':'smf',[98 100]  [Rules]  1 0, 1 (1) : 1  2 5, 1 (1) : 2  3 1, 2 (1) : 1  3 2, 1 (1) : 1  3 3, 1 (1) : 1  3 4, 1 (1) : 1  4 1, 2 (1) : 1  4 2, 2 (1) : 1  4 3, 1 (1) : 1  4 4, 1 (1) : 1  5 1, 4 (1) : 1  5 2, 4 (1) : 1  5 3, 3 (1) : 1  5 4, 2 (1) : 1  6 1, 5 (1) : 1  6 2, 4 (1) : 1  6 3, 4 (1) : 1  6 4, 3 (1) : 1 |

*A03.01 – FIS1 – Light Level Check*

|  |
| --- |
| [System]  Name='FIS2\_Testing'  Type='mamdani'  Version=2.0  NumInputs=2  NumOutputs=1  NumRules=20  AndMethod='prod'  OrMethod='max'  ImpMethod='prod'  AggMethod='max'  DefuzzMethod='centroid'  [Input1]  Name='light'  Range=[0 100]  NumMFs=5  MF1='no\_light':'trapmf',[0 0 0 1]  MF2='low\_light':'pimf',[0 0.5 30 55]  MF3='moderate\_light':'gaussmf',[15 55]  MF4='high\_light':'pimf',[60 85 99 100]  MF5='complete\_light':'smf',[98 100]  [Input2]  Name='alt\_heat'  Range=[-5 30]  NumMFs=4  MF1='none':'trapmf',[-1 -0.5 0.5 1]  MF2='cold\_current':'gbellmf',[4.5 12 -5]  MF3='warm\_current':'pimf',[0 1 10 20]  MF4='thermal\_vents':'smf',[10 20]  [Output1]  Name='heat'  Range=[-5 50]  NumMFs=7  MF1='freezing':'trapmf',[-5 -5 0 0]  MF2='cold':'trapmf',[0 0.5 2 3]  MF3='cool':'pimf',[1 3 6 11]  MF4='temperate':'gbellmf',[6.6 4 13]  MF5='warm':'gbellmf',[7 4 24]  MF6='hot':'gbellmf',[5 4 34]  MF7='very\_hot':'smf',[35 40]  [Rules]  1 1, 2 (1) : 1  1 2, 2 (1) : 1  1 3, 3 (1) : 1  1 4, 5 (1) : 1  2 1, 2 (1) : 1  2 2, 2 (1) : 1  2 3, 3 (1) : 1  2 4, 6 (1) : 1  3 1, 3 (1) : 1  3 2, 2 (1) : 1  3 3, 4 (1) : 1  3 4, 6 (1) : 1  4 1, 4 (1) : 1  4 2, 3 (1) : 1  4 3, 5 (1) : 1  4 4, 7 (1) : 1  5 1, 5 (1) : 1  5 2, 4 (1) : 1  5 3, 6 (1) : 1  5 4, 7 (1) : 1 |

*A03.02 – FIS2 – Heat Level Check*

|  |
| --- |
| [System]  Name='FIS3\_Testing'  Type='mamdani'  Version=2.0  NumInputs=2  NumOutputs=2  NumRules=37  AndMethod='min'  OrMethod='max'  ImpMethod='prod'  AggMethod='max'  DefuzzMethod='mom'  [Input1]  Name='depth'  Range=[-600 0]  NumMFs=6  MF1='trenches':'zmf',[-450 -400]  MF2='abyss':'gbellmf',[100 5 -350]  MF3='midnight':'gbellmf',[70 5 -200]  MF4='twilight':'gbellmf',[45 5 -95]  MF5='sunlight':'gbellmf',[30 5 -30]  MF6='surface':'smf',[-20 0]  [Input2]  Name='heat'  Range=[-5 50]  NumMFs=7  MF1='freezing':'trapmf',[-5 -5 0 0]  MF2='cold':'trapmf',[0 0.5 4.5 6]  MF3='cool':'pimf',[4 6 9 12]  MF4='temperate':'gbellmf',[5 4 15]  MF5='warm':'gbellmf',[6 4 24]  MF6='hot':'gbellmf',[5 4 34]  MF7='very\_hot':'smf',[35 40]  [Output1]  Name='energy'  Range=[0 1000]  NumMFs=5  MF1='barren':'zmf',[10 100]  MF2='sparse':'gbellmf',[135 4 200]  MF3='moderate':'gbellmf',[150 4 480]  MF4='abundant':'gbellmf',[145 4 760]  MF5='eden':'smf',[850 950]  [Output2]  Name='oxygen'  Range=[0 100]  NumMFs=6  MF1='death\_zone':'zmf',[0.25 1]  MF2='very\_low':'gbellmf',[12 2.5 10]  MF3='low':'gbellmf',[14 3 30]  MF4='moderate':'gbellmf',[14 3 55]  MF5='high':'gbellmf',[14 3 78]  MF6='very\_high':'smf',[83 92]  [Rules]  0 1, 1 1 (1) : 1  1 2, 5 5 (1) : 1  1 3, 4 5 (1) : 1  1 4, 4 4 (1) : 1  1 5, 3 4 (1) : 1  1 6, 3 3 (1) : 1  1 7, 2 3 (1) : 1  2 2, 2 3 (1) : 1  2 3, 2 3 (1) : 1  2 4, 1 2 (1) : 1  2 5, 1 2 (1) : 1  2 6, 1 2 (1) : 1  2 7, 1 1 (1) : 1  3 2, 3 5 (1) : 1  3 3, 3 4 (1) : 1  3 4, 2 4 (1) : 1  3 5, 2 3 (1) : 1  3 6, 1 2 (1) : 1  3 7, 1 2 (1) : 1  4 2, 4 5 (1) : 1  4 3, 4 5 (1) : 1  4 4, 3 4 (1) : 1  4 5, 3 4 (1) : 1  4 6, 2 3 (1) : 1  4 7, 1 3 (1) : 1  5 2, 5 6 (1) : 1  5 3, 4 5 (1) : 1  5 4, 4 5 (1) : 1  5 5, 3 4 (1) : 1  5 6, 3 3 (1) : 1  5 7, 2 3 (1) : 1  6 2, 5 6 (1) : 1  6 3, 5 6 (1) : 1  6 4, 4 5 (1) : 1  6 5, 4 4 (1) : 1  6 6, 3 3 (1) : 1  6 7, 2 3 (1) : 1 |

*A03.03 – FIS3 – Energy & Oxygen Check*

|  |
| --- |
| [System]  Name='FIS4\_Testing'  Type='mamdani'  Version=2.0  NumInputs=2  NumOutputs=1  NumRules=26  AndMethod='prod'  OrMethod='max'  ImpMethod='prod'  AggMethod='max'  DefuzzMethod='lom'  [Input1]  Name='energy'  Range=[0 1000]  NumMFs=5  MF1='barren':'zmf',[10 100]  MF2='sparse':'gbellmf',[135 4 200]  MF3='moderate':'gbellmf',[150 4 480]  MF4='abundant':'gbellmf',[145 4 760]  MF5='eden':'smf',[850 950]  [Input2]  Name='oxygen'  Range=[0 100]  NumMFs=6  MF1='death\_zone':'zmf',[0.25 1]  MF2='very\_low':'gbellmf',[12 2.5 10]  MF3='low':'gbellmf',[14 3 30]  MF4='moderate':'gbellmf',[14 3 55]  MF5='high':'gbellmf',[14 3 78]  MF6='very\_high':'smf',[83 92]  [Output1]  Name='survive\_score'  Range=[-100 100]  NumMFs=7  MF1='impossible':'zmf',[-98 -95]  MF2='very\_low':'gaussmf',[25 -80]  MF3='low':'gaussmf',[25 -40]  MF4='possible':'gaussmf',[25 0]  MF5='high':'gaussmf',[25 40]  MF6='very\_high':'gaussmf',[25 80]  MF7='guaranteed':'smf',[95 98]  [Rules]  0 1, 1 (1) : 1  1 2, 2 (1) : 1  1 3, 2 (1) : 1  1 4, 2 (1) : 1  1 5, 2 (1) : 1  1 6, 2 (1) : 1  2 2, 2 (1) : 1  2 3, 3 (1) : 1  2 4, 3 (1) : 1  2 5, 4 (1) : 1  2 6, 4 (1) : 1  3 2, 3 (1) : 1  3 3, 4 (1) : 1  3 4, 4 (1) : 1  3 5, 5 (1) : 1  3 6, 5 (1) : 1  4 2, 4 (1) : 1  4 3, 5 (1) : 1  4 4, 6 (1) : 1  4 5, 6 (1) : 1  4 6, 7 (1) : 1  5 2, 5 (1) : 1  5 3, 6 (1) : 1  5 4, 6 (1) : 1  5 5, 7 (1) : 1  5 6, 7 (1) : 1 |

*A03.04 – FIS4 – Survivability Check*

|  |
| --- |
| [System]  Name='FIS5\_Testing'  Type='mamdani'  Version=2.0  NumInputs=2  NumOutputs=1  NumRules=25  AndMethod='min'  OrMethod='max'  ImpMethod='min'  AggMethod='max'  DefuzzMethod='mom'  [Input1]  Name='popullant'  Range=[0 100]  NumMFs=5  MF1='no\_pollution':'zmf',[0 3]  MF2='low\_pollution':'gbellmf',[17.5 3 15]  MF3='moderate\_pollution':'gaussmf',[15 50]  MF4='high\_pollution':'gbellmf',[17.5 3 85]  MF5='complete\_pollution':'smf',[95 100]  [Input2]  Name='survive\_score'  Range=[-100 100]  NumMFs=7  MF1='impossible':'zmf',[-98 -95]  MF2='very\_low':'gaussmf',[25 -80]  MF3='low':'gaussmf',[25 -40]  MF4='possible':'gaussmf',[25 0]  MF5='high':'gaussmf',[25 40]  MF6='very\_high':'gaussmf',[25 80]  MF7='guaranteed':'smf',[95 98]  [Output1]  Name='survive\_score'  Range=[-100 100]  NumMFs=7  MF1='impossible':'zmf',[-98 -95]  MF2='very\_low':'gaussmf',[25 -80]  MF3='low':'gaussmf',[25 -40]  MF4='possible':'gaussmf',[25 0]  MF5='high':'gaussmf',[25 40]  MF6='very\_high':'gaussmf',[25 80]  MF7='guaranteed':'smf',[95 98]  [Rules]  5 1, 1 (1) : 2  1 2, 2 (1) : 1  1 3, 3 (1) : 1  1 4, 4 (1) : 1  1 5, 5 (1) : 1  1 6, 6 (1) : 1  1 7, 7 (1) : 1  2 2, 1 (1) : 1  2 3, 2 (1) : 1  2 4, 3 (1) : 1  2 5, 4 (1) : 1  2 6, 5 (1) : 1  2 7, 6 (1) : 1  3 2, 1 (1) : 1  3 3, 1 (1) : 1  3 4, 2 (1) : 1  3 5, 3 (1) : 1  3 6, 4 (1) : 1  3 7, 5 (1) : 1  4 2, 1 (1) : 1  4 3, 1 (1) : 1  4 4, 1 (1) : 1  4 5, 2 (1) : 1  4 6, 3 (1) : 1  4 7, 4 (1) : 1 |

*A03.05 – FIS5 – Final Check*

***A04: Fuzzy Test Data***

**FIS 1 – Test Data Results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In1::Depth(m)** | **In2::Pollution(%)** | **Expected** | **Centroid** | **MoM** | **LoM** | **SoM** |
| 0 | 0 | **100** | 84.8 | 100 | 100 | 100 |
| 0 | 20 | **90-95** | 74.5 | 92 | 99 | 85 |
| 0 | 50 | **85-95** | 65.6 | 92 | 99 | 85 |
| 0 | 100 | **0** | 46.0 | 0 | 0 | 0 |
| -40 | 0 | **>90** | 74.7 | 92 | 99 | 85 |
| -40 | 20 | **>90** | 72.9 | 92 | 99 | 85 |
| -40 | 60 | **45-55** | 53.8 | 55 | 68 | 42 |
| -40 | 100 | **0** | 22.5 | 0 | 0 | 0 |
| -120 | 0 | **15-25** | 21.7 | 15.5 | 30 | 1 |
| -120 | 20 | **<20** | 21.7 | 15.5 | 30 | 1 |
| -120 | 60 | **0** | 5.2 | 0 | 0 | 0 |
| -120 | 100 | **0** | 0.2 | 0 | 0 | 0 |
| -250 | 0 | **<20** | 21.6 | 17 | 33 | 1 |
| -250 | 20 | **0** | 0.0 | 0 | 0 | 0 |
| -250 | 60 | **0** | 0.0 | 0 | 0 | 0 |
| -250 | 100 | **0** | 0.0 | 0 | 0 | 0 |
| -600 | 0 | **0** | 0.0 | 0 | 0 | 0 |
| -600 | 20 | **0** | 0.0 | 0 | 0 | 0 |
| -600 | 60 | **0** | 0.0 | 0 | 0 | 0 |
| -600 | 100 | **0** | 0.0 | 0 | 0 | 0 |

*A04.01 – FIS1 Test Data – Defuzzification Methods*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **>>>MoM** | **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 99.0 |
| 92.0 | 92.0 | 92.0 | 92.0 | 92.0 | 90.0 |
| 92.0 | 92.0 | 92.0 | 92.0 | 92.0 | 81.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 40.5 |
| 92.0 | 92.0 | 92.0 | 92.0 | 92.0 | 92.0 |
| 92.0 | 92.0 | 92.0 | 92.0 | 92.0 | 90.0 |
| 55.0 | 55.0 | 55.0 | 55.0 | 55.0 | 45.5 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 15.5 | 15.5 | 15.5 | 15.5 | 15.5 | 15.5 |
| 15.5 | 16.0 | 15.5 | 15.5 | 15.5 | 15.5 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 17.0 | 17.0 | 17.0 | 15.5 | 17.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
|  | **Min** better | | **Prod** better | | **Max** better | |

*A04.02 – FIS1 Test Data – Other Methods*

|  |  |  |  |
| --- | --- | --- | --- |
| **Final Output** | | **MoM** | **Expected** |
| **100.0** | **%** | 100 | **100** |
| **92.0** | **%** | 92 | **90-95** |
| **92.0** | **%** | 92 | **85-95** |
| **0.0** | **%** | 0 | **0** |
| **92.0** | **%** | 92 | **>90** |
| **92.0** | **%** | 92 | **>90** |
| **55.0** | **%** | 55 | **45-55** |
| **0.0** | **%** | 0 | **0** |
| **15.5** | **%** | 15.5 | **15-25** |
| **15.5** | **%** | 15.5 | **<20** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **15.5** | **%** | 17 | **<20** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |
| **0.0** | **%** | 0 | **0** |

*A04.03 – FIS1 Test Data – Final Output & Comparison*

**FIS 2 – Tests Data Results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In1::Light(%)** | **In2::Alt Heat(c)** | **Expected** | **Centriod** | **MoM** | **LoM** | **SoM** |
| 100 | 0 | **20-25** | 19.0 | 24.2 | 24.2 | 24.2 |
| 90 | -4 | **12-16** | 12.3 | 13.2 | 13.2 | 13.2 |
| 80 | 5 | **18-24** | 19.4 | 24.2 | 29.7 | 18.7 |
| 80 | 20 | **>40** | 40.0 | 44.2 | 50.0 | 38.5 |
| 70 | 0 | **6-10** | 9.6 | 4.9 | 7.1 | 2.7 |
| 70 | 15 | **22-28** | 26.7 | 22.2 | 39.0 | 6.6 |
| 60 | 10 | **12-16** | 14.0 | 13.2 | 13.2 | 13.2 |
| 60 | 0 | **5-8** | 7.3 | 4.6 | 6.0 | 3.3 |
| 50 | 0 | **4-8** | 5.6 | 4.9 | 7.1 | 2.7 |
| 50 | 3 | **10-14** | 12.3 | 12.9 | 17.6 | 8.2 |
| 40 | -3 | **0-3** | 2.2 | 1.3 | 2.2 | 0.5 |
| 40 | 0 | **2-5** | 4.7 | 1.3 | 2.2 | 0.5 |
| 30 | 5 | **6-10** | 9.2 | 4.6 | 6.6 | 2.7 |
| 30 | 0 | **1-4** | 3.9 | 1.1 | 1.6 | 0.5 |
| 20 | 20 | **>30** | 34.0 | 34.1 | 34.1 | 34.1 |
| 20 | 0 | **1-3** | 2.9 | 1.1 | 1.6 | 0.5 |
| 10 | 8 | **6-9** | 6.4 | 4.6 | 6.6 | 2.7 |
| 10 | -3 | **0-2** | 1.4 | 1.1 | 1.6 | 0.5 |
| 0 | 5 | **3-7** | 5.6 | 4.6 | 6.0 | 3.3 |
| 0 | 30 | **24-28** | 27.3 | 24.2 | 24.2 | 24.2 |
|  |  |  | **BEST** |  |  |  |

*A04.04 – FIS 2 Defuzzification Test Data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| 19.0 | 19.0 | 19.0 | 19.2 | 19.0 | 17.9 |
| 12.3 | 12.3 | 12.3 | 12.4 | 12.3 | 12.3 |
| 19.4 | 19.4 | 19.4 | 19.6 | 19.4 | 19.0 |
| 40.0 | 40.0 | 40.0 | 40.3 | 40.0 | 39.5 |
| 9.6 | 9.6 | 9.6 | 9.1 | 9.6 | 9.3 |
| 26.7 | 25.1 | 26.7 | 27.2 | 26.7 | 26.7 |
| 14.0 | 14.0 | 14.0 | 13.6 | 14.0 | 13.7 |
| 7.3 | 7.3 | 7.3 | 6.7 | 7.3 | 7.6 |
| 5.6 | 5.6 | 5.6 | 5.2 | 5.6 | 5.8 |
| 12.3 | 12.3 | 12.3 | 12.1 | 12.3 | 11.8 |
| 2.2 | 2.2 | 2.2 | 1.7 | 2.2 | 1.9 |
| 4.7 | 4.7 | 4.7 | 4.4 | 4.7 | 4.5 |
| 9.2 | 9.2 | 9.2 | 8.6 | 9.2 | 8.9 |
| 3.9 | 3.9 | 3.9 | 3.4 | 3.9 | 3.6 |
| 34.0 | 34.0 | 34.0 | 34.0 | 34.0 | 34.0 |
| 2.9 | 2.9 | 2.9 | 2.5 | 2.9 | 2.8 |
| 6.4 | 6.4 | 6.4 | 5.9 | 6.4 | 6.4 |
| 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| 5.6 | 5.6 | 5.6 | 5.5 | 5.6 | 5.5 |
| 27.3 | 27.3 | 27.3 | 27.1 | 27.3 | 27.7 |
| **Prod** tiny bit better | | **Prod** slightly better | | **Max** just better | |

*A04.05 – FIS 2 other Methods test data.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Final Output** | | **Cent-Orig** | **Expected** |
| **19.3** | **c** | 19.0 | **22-26** |
| **12.4** | **c** | 12.3 | **12-16** |
| **19.6** | **c** | 19.4 | **18-24** |
| **40.3** | **c** | 40.0 | **>40** |
| **9.1** | **c** | 9.6 | **6-10** |
| **25.6** | **c** | 26.7 | **22-28** |
| **13.6** | **c** | 14.0 | **12-16** |
| **6.7** | **c** | 7.3 | **5-8** |
| **5.2** | **c** | 5.6 | **4-8** |
| **12.1** | **c** | 12.3 | **10-14** |
| **1.7** | **c** | 2.2 | **0-3** |
| **4.4** | **c** | 4.7 | **2-5** |
| **8.6** | **c** | 9.2 | **6-10** |
| **3.4** | **c** | 3.9 | **1-4** |
| **34.0** | **c** | 34.0 | **>30** |
| **2.5** | **c** | 2.9 | **1-3** |
| **5.9** | **c** | 6.4 | **6-9** |
| **1.4** | **c** | 1.4 | **0-2** |
| **5.5** | **c** | 5.6 | **3-7** |
| **27.1** | **c** | 27.3 | **24-28** |

*A04.06 – FIS final output test data*

**FIS 3 – Test Data Results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In1::Depth(m)** | **In2::Heat(c)** | **Exp.1** | **Out1:Centriod** | **Out1:MoM** | **Out1:LoM** | **Out1:SoM** |
| 0 | 5 | **>900** | 805.7 | 955 | 1000 | 910 |
| 0 | 20 | **>700** | 654.8 | 760 | 850 | 670 |
| 0 | 40 | **<300** | 258.6 | 200 | 200 | 200 |
| -10 | 3 | **>900** | 945.9 | 975 | 1000 | 950 |
| -10 | 12 | **>750** | 757.7 | 760 | 840 | 680 |
| -10 | 24 | **500-600** | 577.5 | 480 | 570 | 390 |
| -50 | 4 | **>900** | 821.2 | 975 | 1000 | 950 |
| -50 | 10 | **>750** | 643.6 | 760 | 880 | 640 |
| -50 | 20 | **450-600** | 578.6 | 480 | 580 | 380 |
| -120 | 1 | **>750** | 702.2 | 760 | 820 | 700 |
| -120 | 8 | **>750** | 678.4 | 760 | 820 | 700 |
| -120 | 16 | **450-550** | 431.2 | 480 | 550 | 410 |
| -200 | 3 | **500** | 475.8 | 480 | 480 | 480 |
| -200 | 20 | **<300** | 201.0 | 200 | 290 | 110 |
| -200 | 30 | **<150** | 166.0 | 15 | 30 | 0 |
| -400 | 2 | **<250** | 200.9 | 200 | 250 | 150 |
| -400 | 13 | **<100** | 28.0 | 5 | 10 | 0 |
| -400 | 23 | **<100** | 28.0 | 5 | 10 | 0 |
| -600 | 3 | **>950** | 948.8 | 975 | 1000 | 950 |
| -600 | 28 | **450-550** | 480.1 | 480 | 580 | 380 |

*A04.07 – FIS 3 Output 1 Energy Defuzzification Test Data*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In1::Depth(m)** | **In2::Heat(c)** | **Exp.2** | **Out2:Centriod** | **Out2:MoM** | **Out2:LoM** | **Out2:SoM** |
| 0 | 5 | **>90** | 81.3 | 95 | 100 | 89 |
| 0 | 20 | **>60** | 63.2 | 55 | 63 | 47 |
| 0 | 40 | **<40** | 30.1 | 30 | 30 | 30 |
| -10 | 3 | **>90** | 93.5 | 96 | 100 | 92 |
| -10 | 12 | **>80** | 77.6 | 78 | 85 | 71 |
| -10 | 24 | **<60** | 55.0 | 55 | 62 | 48 |
| -50 | 4 | **>90** | 83.1 | 96 | 100 | 92 |
| -50 | 10 | **>75** | 68.5 | 78 | 89 | 67 |
| -50 | 20 | **55-65** | 63.2 | 55 | 63 | 47 |
| -120 | 1 | **>75** | 77.7 | 78 | 83 | 73 |
| -120 | 8 | **>75** | 73.1 | 78 | 83 | 73 |
| -120 | 16 | **55-65** | 52.4 | 55 | 60 | 50 |
| -200 | 3 | **>80** | 76.4 | 78 | 78 | 78 |
| -200 | 20 | **<35** | 39.3 | 30 | 38 | 22 |
| -200 | 30 | **<25** | 20.8 | 10 | 18 | 2 |
| -400 | 2 | **<35** | 30.1 | 30 | 34 | 26 |
| -400 | 13 | **<20** | 12.0 | 10 | 13 | 7 |
| -400 | 23 | **<20** | 12.0 | 10 | 13 | 7 |
| -600 | 3 | **>80** | 77.7 | 78 | 78 | 78 |
| -600 | 28 | **<35** | 28.7 | 30 | 38 | 22 |

*A04.08 – FIS 3 Output 2 Oxygen Defuzzification Test Data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| 955 | 955 | 955 | 975 | 955 | 910 |
| 760 | 760 | 760 | 760 | 760 | 630 |
| 200 | 200 | 200 | 200 | 200 | 300 |
| 975 | 975 | 975 | 975 | 975 | 950 |
| 760 | 760 | 760 | 760 | 760 | 680 |
| 480 | 480 | 480 | 480 | 480 | 610 |
| 975 | 975 | 975 | 975 | 975 | 930 |
| 760 | 760 | 760 | 760 | 760 | 630 |
| 480 | 480 | 480 | 480 | 480 | 610 |
| 760 | 760 | 760 | 760 | 760 | 660 |
| 760 | 760 | 760 | 760 | 760 | 660 |
| 480 | 480 | 480 | 480 | 480 | 360 |
| 480 | 480 | 480 | 480 | 480 | 420 |
| 200 | 200 | 200 | 200 | 200 | 200 |
| 15 | 15 | 15 | 5 | 15 | 30 |
| 200 | 200 | 200 | 200 | 200 | 200 |
| 5 | 5 | 5 | 5 | 5 | 5 |
| 5 | 5 | 5 | 5 | 5 | 5 |
| 975 | 975 | 975 | 975 | 975 | 950 |
| 480 | 480 | 480 | 480 | 480 | 485 |

*A04.09 – FIS 3 Output 1 Energy Other Methods Test Data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| 95 | 95 | 95 | 96 | 95 | 91 |
| 55 | 55 | 55 | 55 | 55 | 64 |
| 30 | 30 | 30 | 30 | 30 | 30 |
| 96 | 96 | 96 | 96 | 96 | 96 |
| 78 | 78 | 78 | 78 | 78 | 78 |
| 55 | 55 | 55 | 55 | 55 | 55 |
| 96 | 96 | 96 | 96 | 96 | 92 |
| 78 | 78 | 78 | 78 | 78 | 68 |
| 55 | 55 | 55 | 55 | 55 | 64 |
| 78 | 78 | 78 | 78 | 78 | 73 |
| 78 | 78 | 78 | 78 | 78 | 72 |
| 55 | 55 | 55 | 55 | 55 | 50 |
| 78 | 78 | 78 | 78 | 78 | 75 |
| 30 | 30 | 30 | 30 | 30 | 41 |
| 10 | 10 | 10 | 10 | 10 | 17 |
| 30 | 30 | 30 | 30 | 30 | 30 |
| 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 |
| 78 | 78 | 78 | 78 | 78 | 75 |
| 30 | 30 | 30 | 30 | 30 | 24 |

*A04.10 – FIS 3 Output 2 Oxygen Other Methods Test Data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Final Outputs** | | **Out1:MoM** | **Out2:MoM** | **Exp.1** | **Exp.2** |
| **975.0** | **96.0** | 955 | 95 | **>900** | **>90** |
| **760.0** | **55.0** | 760 | 55 | **>700** | **>60** |
| **200.0** | **30.0** | 200 | 30 | **<300** | **<40** |
| **975.0** | **96.0** | 975 | 96 | **>900** | **>90** |
| **760.0** | **78.0** | 760 | 78 | **>750** | **>80** |
| **480.0** | **55.0** | 480 | 55 | **500-600** | **<60** |
| **975.0** | **96.0** | 975 | 96 | **>900** | **>90** |
| **760.0** | **78.0** | 760 | 78 | **>800** | **>80** |
| **480.0** | **55.0** | 480 | 55 | **550-650** | **60-70** |
| **760.0** | **78.0** | 760 | 78 | **>800** | **>80** |
| **760.0** | **78.0** | 760 | 78 | **>750** | **>75** |
| **480.0** | **55.0** | 480 | 55 | **450-550** | **55-65** |
| **480.0** | **78.0** | 480 | 78 | **500** | **>80** |
| **200.0** | **30.0** | 200 | 30 | **<300** | **<35** |
| **5.0** | **10.0** | 15 | 10 | **<150** | **<25** |
| **200.0** | **30.0** | 200 | 30 | **<250** | **<35** |
| **5.0** | **10.0** | 5 | 10 | **<100** | **<20** |
| **5.0** | **10.0** | 5 | 10 | **<100** | **<20** |
| **975.0** | **78.0** | 975 | 78 | **>950** | **>80** |
| **480.0** | **30.0** | 480 | 30 | **450-550** | **<35** |

*A04.11 – FIS 3 Outputs 1 & 2 Energy & Oxygen Final Output Test Data*

**FIS 4 – Test Data Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **In2::Oxygen(%)** | **Expected** | **Centriod** | **MoM** | **LoM** | **SoM** |
| 0 | **-100** | -69.6 | -99 | -100 | -98 |
| 100 | **<-60** | -54.2 | -80 | -90 | -70 |
| 30 | **<-80** | -54.0 | -78 | -100 | -56 |
| 70 | **<-10** | -13.3 | 0 | -10 | 0 |
| 10 | **<-80** | -64.8 | -80 | -80 | -80 |
| 90 | **0** | 1.1 | 0 | -10 | 0 |
| 35 | **<-40** | -31.1 | -40 | -50 | -30 |
| 85 | **>35** | 30.0 | 40 | 52 | 28 |
| 15 | **<-40** | -25.7 | -40 | -42 | -38 |
| 95 | **>35** | 39.3 | 40 | 40 | 40 |
| 20 | **<-20** | -16.5 | 0 | -12 | 0 |
| 70 | **>60** | 28.4 | 40 | 52 | 28 |
| 25 | **>40** | 24.8 | 40 | 42 | 38 |
| 100 | **100** | 60.3 | 99 | 100 | 98 |
| 50 | **>60** | 63.4 | 80 | 80 | 80 |
| 80 | **>90** | 70.8 | 80 | 80 | 80 |
| 5 | **0** | 1.1 | 0 | -2 | 0 |
| 45 | **>85** | 55.3 | 80 | 94 | 66 |
| 70 | **>95** | 68.0 | 99 | 100 | 98 |
| 100 | **100** | 84.8 | 99 | 100 | 98 |

*A04.12 – FIS 4 Defuzzification Test Data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| -100 | -100 | -100 | -100 | -100 | -98 |
| -90 | -90 | -90 | -80 | -90 | -70 |
| -100 | -100 | -100 | -80 | -100 | -56 |
| -10 | -12 | -10 | 0 | -10 | -10 |
| -80 | -80 | -80 | -80 | -80 | -80 |
| -10 | -10 | -10 | 0 | -10 | -10 |
| -50 | -50 | -50 | -40 | -50 | -30 |
| 52 | 54 | 52 | 40 | 52 | 28 |
| -42 | -42 | -42 | -40 | -42 | -34 |
| 40 | 40 | 40 | 40 | 40 | 40 |
| -12 | -12 | -12 | 0 | -12 | -18 |
| 52 | 54 | 52 | 40 | 52 | 42 |
| 42 | 42 | 42 | 40 | 42 | 38 |
| 100 | 100 | 100 | 100 | 100 | 98 |
| 80 | 80 | 80 | 80 | 80 | 78 |
| 80 | 80 | 80 | 80 | 80 | 80 |
| -2 | -4 | -2 | 0 | -2 | -2 |
| 94 | 100 | 94 | 80 | 94 | 66 |
| 100 | 100 | 100 | 100 | 100 | 98 |
| 100 | 100 | 100 | 100 | 100 | 98 |
| **Prod** better | | **Prod** better, just | | **Max** better | |

*A04.13 – FIS 4 other Methods Test Data*

|  |  |  |
| --- | --- | --- |
| **Final Output** | **LoM** | **Expected** |
| **-100.0** | -100 | **-100** |
| **-80.0** | -90 | **<-60** |
| **-80.0** | -100 | **<-80** |
| **0.0** | -10 | **<-40** |
| **-80.0** | -80 | **<-80** |
| **0.0** | -10 | **0** |
| **-40.0** | -50 | **<-40** |
| **40.0** | 52 | **>35** |
| **-40.0** | -42 | **<-40** |
| **40.0** | 40 | **>35** |
| **0.0** | -12 | **<-20** |
| **40.0** | 52 | **>60** |
| **40.0** | 42 | **>40** |
| **100.0** | 100 | **100** |
| **80.0** | 80 | **>60** |
| **80.0** | 80 | **>90** |
| **0.0** | -2 | **0** |
| **80.0** | 94 | **>85** |
| **100.0** | 100 | **>95** |
| **100.0** | 100 | **100** |

*A04.14 – FIS 4 final output test data*

**FIS 5 – Test Data Results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In1::Pollution(%)** | **In2::Survive(int)** | **Expected** | **Centriod** | **MoM** | **LoM** | **SoM** |
| 0 | -100 | -100 | -66.6 | -99 | -100 | -98 |
| 0 | -50 | <-50 | -43.6 | -40 | -50 | -30 |
| 0 | 50 | >50 | 21.8 | 40 | 50 | 30 |
| 0 | 100 | 100 | 48.1 | 99 | 100 | 98 |
| 10 | 80 | >50 | 27.4 | 40 | 40 | 40 |
| 20 | -40 | <-80 | -59.2 | -80 | -80 | -80 |
| 25 | 0 | <-40 | -30.9 | -40 | -46 | -34 |
| 30 | 50 | 0 | 1.2 | 0 | -20 | 0 |
| 40 | -60 | <-90 | -55.3 | -99 | -100 | -98 |
| 50 | 45 | >-40 | -27.0 | -40 | -44 | -36 |
| 55 | 0 | <-80 | -58.7 | -80 | -88 | -72 |
| 60 | 80 | <0 | -11.2 | 0 | -16 | 0 |
| 70 | -20 | <-90 | -61.7 | -99 | -100 | -98 |
| 75 | 0 | <-60 | -48.4 | -99 | -100 | -98 |
| 80 | 80 | <-40 | -35.5 | -40 | -40 | -40 |
| 95 | 45 | <-95 | -56.6 | -80 | -86 | -74 |
| 100 | -100 | -100 | -98.0 | -99 | -100 | -98 |
| 100 | -20 | -100 | -73.0 | -99 | -100 | -98 |
| 100 | 30 | -100 | -62.2 | -99 | -100 | -98 |
| 100 | 100 | -100 | -23.1 | -99 | -100 | -98 |

*Figure 49 – FIS 5 defuzzification test data*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **And:Min** | **And:Prod** | **Imp:Min** | **Imp:Prod** | **Agg:Max** | **Agg:Sum** |
| -99 | -99 | -99 | -99 | -99 | -99 |
| -40 | -40 | -40 | -40 | -40 | -50 |
| 40 | 40 | 40 | 40 | 40 | 30 |
| 99 | 99 | 99 | 99 | 99 | 98 |
| 40 | 40 | 40 | 40 | 40 | 40 |
| -80 | -80 | -80 | -80 | -80 | -96 |
| -40 | -40 | -40 | -40 | -40 | -40 |
| 0 | 0 | 0 | 0 | 0 | 10 |
| -99 | -99 | -99 | -99 | -99 | -99 |
| -40 | -40 | -40 | -40 | -40 | -34 |
| -80 | -80 | -80 | -80 | -80 | -72 |
| 0 | 0 | 0 | 0 | 0 | -16 |
| -99 | -99 | -99 | -99 | -99 | -98 |
| -99 | -99 | -99 | -99 | -99 | -98 |
| -40 | -40 | -40 | -40 | -40 | -40 |
| -80 | -80 | -80 | -80 | -80 | -74 |
| -99 | -99 | -99 | -99 | -99 | -99 |
| -99 | -99 | -99 | -99 | -99 | -98 |
| -99 | -99 | -99 | -99 | -99 | -98 |
| -99 | -99 | -99 | -99 | -99 | -20 |
| no difference | | no difference | | **Max** better | |

*A04.15 – FIS other methods test data*

|  |  |  |
| --- | --- | --- |
| **Final Output** | **MoM** | **Expected** |
| **-99.0** | -99 | -100 |
| **-40.0** | -40 | <-50 |
| **40.0** | 40 | >50 |
| **99.0** | 99 | 100 |
| **40.0** | 40 | >50 |
| **-80.0** | -80 | <-80 |
| **-40.0** | -40 | <-40 |
| **0.0** | 0 | 0 |
| **-99.0** | -99 | <-90 |
| **-40.0** | -40 | >-40 |
| **-80.0** | -80 | <-80 |
| **0.0** | 0 | <0 |
| **-99.0** | -99 | <-90 |
| **-99.0** | -99 | <-60 |
| **-40.0** | -40 | <-40 |
| **-80.0** | -80 | <-95 |
| **-99.0** | -99 | -100 |
| **-99.0** | -99 | -100 |
| **-99.0** | -99 | -100 |
| **-99.0** | -99 | -100 |

*A04.16 – FIS final output test data.*

***TESTING DATA***

***B01: Population Test Tables***

*B01.01 – Seed Population Pool Size Testing – Number of failures - Ran 1000 times on each.*

*B01.02 – Seed Population Pool Size Testing – Failure % - Ran 1000 times on each.*

***B02: Population Test Data***

|  |  |  |
| --- | --- | --- |
| *Test ran 1000 times on each* | | |
| **Pool Size** | **Num. Fails** | **Fail %** |
| 1000 | 789 | 78.90 |
| 2500 | 426 | 17.04 |
| 5000 | 277 | 5.54 |
| 7500 | 131 | 1.75 |
| 10000 | 94 | 0.94 |
| 12500 | 47 | 0.38 |
| 15000 | 21 | 0.14 |
| 17500 | 17 | 0.10 |
| 20000 | 18 | 0.09 |
| 22500 | 12 | 0.05 |
| 25000 | 11 | 0.04 |

*B02.01 – Data for seed population testing – ran 1000 times on each value.*

***B03: Mutation Test Tables***

*B03.01 – Mutation Intensity Value Testing – Largest Single Mutation*

*B03.02 – Mutation Intensity Value Testing – Averaged Mutaion Value*

*B03.03 – Mutation Intensity Value Testing - Chance of Mutation larger than 1% of start value.*

*B03.04 – Mutation Gaussian vs Uniform – Largest single mutation value.*

*B03.05 – Mutation Gaussian vs Uniform - Averaged Mutation Value*

*B03.06 – Mutation Gaussian vs Uniform -Chance of mutation greater than 1% of start value.*

***B04: Mutation Test Data***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Test ran 1,000,000 times on each intensity value*** | | | |
| **Intensity** | **Largest** | **Average** | **% changes over 1% of initial value** |
| 0.05 | 0.21 | 0.001 | 0.000 |
| 0.1 | 0.43 | 0.002 | 0.000 |
| 0.15 | 0.62 | 0.003 | 0.000 |
| 0.2 | 0.9 | 0.004 | 0.000 |
| 0.25 | 0.98 | 0.005 | 0.000 |
| 0.3 | 1.29 | 0.006 | 0.000 |
| 0.35 | 1.39 | 0.007 | 0.000 |
| 0.4 | 1.98 | 0.008 | 0.000 |
| 0.45 | 2.14 | 0.009 | 0.001 |
| 0.5 | 2.33 | 0.010 | 0.001 |
| 0.55 | 2.24 | 0.011 | 0.002 |
| 0.6 | 2.43 | 0.012 | 0.002 |
| 0.65 | 2.83 | 0.013 | 0.003 |
| 0.7 | 2.89 | 0.014 | 0.004 |
| 0.75 | 2.96 | 0.015 | 0.005 |
| 0.8 | 3.3 | 0.016 | 0.005 |
| 0.85 | 3.29 | 0.017 | 0.006 |
| 0.9 | 3.92 | 0.018 | 0.007 |
| 0.95 | 4.15 | 0.019 | 0.007 |
| 1 | 4.1 | 0.020 | 0.008 |
| 1.25 | 5.35 | 0.025 | 0.011 |
| 1.5 | 5.85 | 0.030 | 0.013 |
| 1.75 | 7.41 | 0.034 | 0.014 |
| 2 | 8.48 | 0.040 | 0.015 |
| 2.25 | 8.55 | 0.045 | 0.016 |
| 2.5 | 10.16 | 0.050 | 0.017 |
| 2.75 | 10.81 | 0.055 | 0.018 |
| 3 | 12.43 | 0.061 | 0.019 |
| 3.25 | 13.74 | 0.064 | 0.019 |
| 3.5 | 16.02 | 0.070 | 0.020 |
| 3.75 | 15.91 | 0.075 | 0.020 |
| 4 | 16.21 | 0.080 | 0.020 |
| 4.25 | 17.93 | 0.085 | 0.021 |
| 4.5 | 18.07 | 0.090 | 0.021 |
| 4.75 | 21.05 | 0.094 | 0.021 |
| 5 | 20.89 | 0.100 | 0.021 |

*B04.01 – Data mutation intensity value testing – ran 1,000,000 on each value.*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Test ran 1,000,000 times on each intensity value*** | | | |
| **GAUSSIAN** | | | |
| **Intensity** | **Largest** | **Average** | **% changes over 1% of initial value** |
| 1 | 4.1 | 0.020 | 0.008 |
| 1.25 | 5.35 | 0.025 | 0.011 |
| 1.5 | 5.85 | 0.030 | 0.013 |
| 1.75 | 7.41 | 0.034 | 0.014 |
| 2 | 8.48 | 0.040 | 0.015 |
| 2.25 | 8.55 | 0.045 | 0.016 |
| 2.5 | 10.16 | 0.050 | 0.017 |
| 2.75 | 10.81 | 0.055 | 0.018 |
| 3 | 12.43 | 0.061 | 0.019 |
| 3.25 | 13.74 | 0.064 | 0.019 |
| 3.5 | 16.02 | 0.070 | 0.020 |
| 3.75 | 15.91 | 0.075 | 0.020 |
| 4 | 16.21 | 0.080 | 0.020 |
| 4.25 | 17.93 | 0.085 | 0.021 |
| 4.5 | 18.07 | 0.090 | 0.021 |
| 4.75 | 21.05 | 0.094 | 0.021 |
| 5 | 20.89 | 0.100 | 0.021 |

*B04.02 – Data mutation gaussian vs uniform – gaussian data – ran 1,000,000 on each value.*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Test ran 1,000,000 times on each intensity value*** | | | |
| **UNIFORM** | | | |
| **Intensity** | **Largest** | **Average** | **% changes over 1% of initial value** |
| 1 | 2.4 | 0.036 | 0.014 |
| 1.25 | 2.74 | 0.045 | 0.019 |
| 1.5 | 2.85 | 0.054 | 0.023 |
| 1.75 | 3.47 | 0.062 | 0.025 |
| 2 | 4.01 | 0.071 | 0.028 |
| 2.25 | 4.22 | 0.080 | 0.029 |
| 2.5 | 5.98 | 0.090 | 0.031 |
| 2.75 | 5.81 | 0.098 | 0.032 |
| 3 | 6.4 | 0.108 | 0.033 |
| 3.25 | 6.9 | 0.115 | 0.034 |
| 3.5 | 7.34 | 0.125 | 0.035 |
| 3.75 | 7.98 | 0.134 | 0.035 |
| 4 | 8.2 | 0.143 | 0.036 |
| 4.25 | 8.22 | 0.152 | 0.037 |
| 4.5 | 8.75 | 0.162 | 0.037 |
| 4.75 | 9.08 | 0.168 | 0.037 |
| 5 | 9.43 | 0.180 | 0.038 |

*B04.03 – Data mutation gaussian vs uniform – uniform data – ran 1,000,000 on each value.*

***B05: ‘Bacteria’ Stage Test Tables***

*B05.01 – ‘Bacteria’ Testing – Test Set 1 – Exponential Growth*

*B05.02 – ‘Bacteria’ Testing – Test Set 2 – Misleading figures suggesting variety.*

*B05.03 – ‘Bacteria’ Testing – Test Set 2 – Deaths by failed fitness tests*

*B05.04 – ‘Bacteria’ Testing – Test Set 2 - % of populations failing fitness tests*

*B05.05 – ‘Bacteria’ Testing – Test Set 3 – more typical results.*

***B06: ‘Bacteria’ Stage Test Data***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TESTING ONE** | | | | |
| **ID** | **346** | **1932** | **4245** | **8009** |
| **0** | 1 | 1 | 1 | 1 |
| **1** | 2 | 2 | 2 | 2 |
| **2** | 4 | 4 | 4 | 4 |
| **3** | 8 | 8 | 7 | 8 |
| **4** | 16 | 16 | 14 | 16 |
| **5** | 32 | 30 | 28 | 32 |
| **6** | 64 | 60 | 56 | 63 |
| **7** | 128 | 120 | 112 | 126 |
| **8** | 256 | 239 | 224 | 251 |
| **9** | 507 | 477 | 448 | 502 |
| **10** | 1014 | 949 | 896 | 997 |
| **11** | 2021 | 1872 | 1790 | 1993 |
| **12** | 3972 | 3738 | 3572 | 3985 |
| **13** | 7892 | 7381 | 7093 | 7967 |
| **14** | 15430 | 14228 | 14026 | 15921 |
| **15** | 30219 | 28444 | 27899 | 31807 |
| **16** | 59937 | 56505 | 55198 | 63490 |
| **17** | 117218 | 112877 | 101693 | 126892 |
| **18** | 229872 | 244993 | 191952 | 253581 |
| **19** | 432091 | 480422 | 348732 | 506744 |
| **20** | 849903 | 948376 | 623646 | 1011301 |
| **21** | 1693491 | 1872752 | 1172492 | 2012342 |
| **22** | 3210512 | 3690503 | 2298372 | 4018846 |
| **23** | 6382541 | 7337006 | 4328193 | 8031004 |
| **24** | 12592001 | 14551012 | 8256863 | 16045921 |
| **25** | 25104774 | 28892042 | 16135462 | 31989103 |

*B06.01 – Bacteria data – setup 1 – exponential growth.*

|  |  |  |  |
| --- | --- | --- | --- |
| **TESTING TWO** | | | |
| **ID** | **2491** | **2760** | **9112** |
| **0** | 12 | 12 | 12 |
| **1** | 24 | 23 | 24 |
| **2** | 48 | 41 | 48 |
| **3** | 96 | 72 | 96 |
| **4** | 192 | 132 | 191 |
| **5** | 384 | 231 | 380 |
| **6** | 768 | 407 | 752 |
| **7** | 1536 | 749 | 1491 |
| **8** | 3072 | 1372 | 2912 |
| **9** | 6138 | 2410 | 5724 |
| **10** | 12243 | 4387 | 11338 |
| **11** | 24431 | 7609 | 22363 |
| **12** | 48701 | 13371 | 44260 |
| **13** | 97350 | 22427 | 88201 |
| **14** | 193654 | 37548 | 173204 |
| **15** | 386812 | 64906 | 340480 |
| **16** | 773187 | 119281 | 642844 |
| **17** | 1544743 | 213462 | 1258344 |
| **18** | 3083845 | 397429 | 2478866 |
| **19** | 6159906 | 720006 | 4890581 |
| **20** | 12317188 | 1331716 | 9601646 |

*B06.02 – Bacteria data – setup 2 – strange misleading figures.*

|  |  |  |  |
| --- | --- | --- | --- |
| **TESTING TWO-B (Failed Fitness Tests)** | | | |
| **ID** | **2491** | **2760** | **9112** |
| **0** | 0 | 0 | 0 |
| **1** | 0 | 1 | 0 |
| **2** | 0 | 5 | 0 |
| **3** | 0 | 10 | 0 |
| **4** | 0 | 12 | 1 |
| **5** | 0 | 33 | 2 |
| **6** | 0 | 55 | 8 |
| **7** | 0 | 65 | 13 |
| **8** | 0 | 126 | 70 |
| **9** | 6 | 334 | 100 |
| **10** | 33 | 433 | 110 |
| **11** | 55 | 1165 | 313 |
| **12** | 161 | 1847 | 466 |
| **13** | 52 | 4315 | 319 |
| **14** | 1046 | 7306 | 3198 |
| **15** | 496 | 10190 | 5928 |
| **16** | 437 | 10531 | 38116 |
| **17** | 1631 | 25100 | 27344 |
| **18** | 5641 | 29495 | 37822 |
| **19** | 7784 | 74852 | 67151 |
| **20** | 2624 | 108296 | 179516 |

*B06.03 – Bacteria data – setup 2 – failed fitness tests, numbers per cycle*

|  |  |  |  |
| --- | --- | --- | --- |
| **TESTING TWO-C (% Pop Failed Fitness Test)** | | | |
| **ID** | **2491** | **2760** | **9112** |
| **0** | 0.00 | 0.00 | 0.00 |
| **1** | 0.00 | 4.35 | 0.00 |
| **2** | 0.00 | 12.20 | 0.00 |
| **3** | 0.00 | 13.89 | 0.00 |
| **4** | 0.00 | 9.09 | 0.52 |
| **5** | 0.00 | 14.29 | 0.53 |
| **6** | 0.00 | 13.51 | 1.06 |
| **7** | 0.00 | 8.68 | 0.87 |
| **8** | 0.00 | 9.18 | 2.40 |
| **9** | 0.10 | 13.86 | 1.75 |
| **10** | 0.27 | 9.87 | 0.97 |
| **11** | 0.23 | 15.31 | 1.40 |
| **12** | 0.33 | 13.81 | 1.05 |
| **13** | 0.05 | 19.24 | 0.36 |
| **14** | 0.54 | 19.46 | 1.85 |
| **15** | 0.13 | 15.70 | 1.74 |
| **16** | 0.06 | 8.83 | 5.93 |
| **17** | 0.11 | 11.76 | 2.17 |
| **18** | 0.18 | 7.42 | 1.53 |
| **19** | 0.13 | 10.40 | 1.37 |
| **20** | 0.02 | 8.13 | 1.87 |

*B06.04 – Bacteria data – setup 2 – % of populations that failed fitness tests.*

|  |  |  |  |
| --- | --- | --- | --- |
| **TESTING THREE** | | | |
| **ID** | **307** | **5821** | **6204** |
| **0** | 12 | 12 | 12 |
| **1** | 24 | 24 | 24 |
| **2** | 48 | 48 | 48 |
| **3** | 96 | 96 | 96 |
| **4** | 192 | 192 | 191 |
| **5** | 383 | 384 | 379 |
| **6** | 754 | 768 | 749 |
| **7** | 1507 | 1536 | 1496 |
| **8** | 3003 | 3068 | 2987 |
| **9** | 5987 | 6133 | 5964 |
| **10** | 11904 | 12227 | 11828 |
| **11** | 23679 | 24389 | 23565 |
| **12** | 46908 | 48728 | 46789 |
| **13** | 93616 | 97897 | 92043 |
| **14** | 187003 | 194989 | 181970 |
| **15** | 373820 | 388278 | 361490 |
| **16** | 746677 | 774101 | 718809 |
| **17** | 1491543 | 1532202 | 1432618 |
| **18** | 2971086 | 3034744 | 2856623 |
| **19** | 5931721 | 6047481 | 5698624 |
| **20** | 11846211 | 12013629 | 11379952 |

*B06.05 – Bacteria data – setup 3 – more exponential growth.*

***B07: Selection Test Tables***

*B07.01 – Selection Testing – declining populations.*

***B08: Selection Test Data***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Starting Pop. 100 || Hard-coded to 4 offspring per cycle* | | | | | |
| % to reproduce | 1 | 2 | 3 | 4 | 5 |
| 5 | 8 | 0 | 0 | 0 | 0 |
| 10 | 20 | 4 | 0 | 0 | 0 |
| 15 | 28 | 8 | 0 | 0 | 0 |
| 20 | 40 | 16 | 4 | 0 | 0 |
| 25 | 48 | 24 | 12 | 4 | 0 |
| 30 | 60 | 36 | 20 | 12 | 4 |
| 35 | 68 | 44 | 28 | 16 | 8 |
| 40 | 80 | 64 | 48 | 36 | 28 |

*B08.01 – Selection Data – declining population test data.*

***B09: ‘Two Parent’ Stage Test Tables***

*B09.01 – ‘Two Parent’ Stage Testing – imbalanced birth/death ratio.*

*B09.02 – ‘Two Parent’ Stage Testing – birth/death ratio rebalance with life spans set to 2, no fitness tests applied.*

*B09.03 – ‘Two Parent’ Stage Testing – birth/death ratio rebalance with life spans set to 2, yes, fitness tests applied.*

*B09.04 – ‘Two Parent’ Stage Testing – comparison of approaches.*

***B10: ‘Two Parent’ Stage Test Data***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | *Live for 1 life cycles || 30% parent selection || 4 offspring per cycle ||* ***NO*** *FITNESS TESTS* | | | | | | | | | | | | | Creature | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | Total | 100 | 160 | 96 | 56 | 32 | 16 | 4 | 0 | 0 | 0 | 0 | | Births |  | 60 | 36 | 20 | 12 | 4 | 0 | 0 | 0 | 0 | 0 | | Deaths | 0 | 100 | 60 | 36 | 20 | 12 | 4 | 0 | 0 | 0 | 0 |   *B10.01 – ‘Two Parent’ Stage Testing – 1 life span, 4 offspring, no fitness tests*  *Live for 2 life cycles || 30% parent selection || 4 offspring per cycle ||* ***NO*** *FITNESS TESTS* | | | | | | | | | | | |
| Creature | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Total | 100 | 160 | 256 | 248 | 300 | 316 | 356 | 388 | 440 | 492 | 556 |
| Births |  | 60 | 96 | 92 | 112 | 112 | 132 | 144 | 164 | 184 | 208 |
| Deaths | 0 | 0 | 100 | 60 | 96 | 92 | 112 | 112 | 132 | 144 | 164 |

*B10.02 – ‘Two Parent’ Stage Testing – 2 life span, 4 offspring, no fitness tests*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Live for 2 life cycles || 30% parent selection || 4 offspring per cycle ||* ***YES*** *FITNESS TESTS* | | | | | | | | | | | |
| Creature | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Total | 100 | 160 | 256 | 240 | 273 | 279 | 291 | 300 | 306 | 320 | 324 |
| Births |  | 60 | 92 | 88 | 100 | 104 | 104 | 112 | 116 | 120 | 120 |
| Deaths | 0 | 2 | 104 | 67 | 98 | 92 | 103 | 110 | 106 | 116 | 120 |
| Old Age | 0 | 0 | 100 | 60 | 92 | 88 | 100 | 104 | 104 | 112 | 116 |
| Failed FT |  | 2 | 4 | 7 | 6 | 4 | 3 | 6 | 2 | 4 | 4 |

*B10.03 – ‘Two Parent’ Stage Testing – 2 life span, 4 offspring, yes fitness tests*

***B11: Environmental Status Test Tables***

*B11.01 – Environmental Status Testing – famines killing off all but very small creatures..*

*B11.02 – Environmental Status Testing – tuning of values, comparisons between set ups.*

***B12: Environmental Status Test Data***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Explosive growth, even with environmental status, closer inspection small creatures still growing in numbers.* | | | | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Weight | 465.96 | 663.66 | 933.66 | 1304.20 | 1849.76 | 2838.35 | 4421.21 | 6952.02 | 9258.12 | 14303.76 |
|  |  |  |  |  |  |  |  |  |  |  |
| V\_LARGE | 8.3 | 5.6 | 3.5 | 1.2 | 0.4 | 0 | 0 | 0 | 0 | 0 |
| LARGE | 12.7 | 10.2 | 6.2 | 3 | 1.1 | 0.2 | 0 | 0 | 0 | 0 |
| MEDIUM | 18.1 | 19.2 | 21.4 | 21.8 | 21.5 | 17.7 | 8.8 | 1.7 | 0 | 0 |
| SMALL | 26.5 | 28.8 | 30.2 | 33.4 | 33.2 | 31.2 | 28.6 | 20.9 | 10.3 | 4.5 |
| V\_SMALL | 34.4 | 36.2 | 38.7 | 41.6 | 43.8 | 50.9 | 62.6 | 77.4 | 89.7 | 95.5 |

*B12.01 – Environmental Status – famine issue – breakdown of surviving creatures.*

*Comparison of different tunings of the competition.h variables.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Setup01 | 404.31 | 634.90 | 892.10 | 1258.20 | 1701.28 | 2543.98 | 3892.74 | 5973.84 | 8720.02 | 12970.67 | 19001.59 | 28449.35 |
| Setup02 | 408.82 | 689.21 | 938.9 | 1301.83 | 1894.85 | 2793.54 | 295.4 | 401.34 | 634.7 | 856.92 | 1201.5 | 1834.61 |
| Setup03 | 389.29 | 599.45 | 819.75 | 1193.7 | 1634.89 | 2480.11 | 982.88 | 1392.69 | 1930.04 | 2873.88 | 1035.21 | 1517.83 |
| Setup04 | 399.1 | 606.79 | 858.29 | 1232.38 | 1490.32 | 1082.73 | 1462.68 | 1010.12 | 1389.98 | 994.07 | 1378.56 | 1058.6 |

*B12.02 – Environmental Status – comparisons of tuning setups.*

***CODE DATA***

***C01: Environmental Creation***

|  |
| --- |
| class EnvironmentCreation  {  public:  EnvironmentCreation(); //!< default constructor.  ~EnvironmentCreation(); //!< default deconstructor.  void environmentCreation(Environment& envir, float energyMin, float energyMax, float tempMin, float tempMax, float oxygenMin, float oxygenMax); //!< generates a randomised environment.  uint32\_t count = 0; //!< count for environments.  private:  inline float setEnergyAvailable(float min, float max) {  energy = genFunc->uniformFloatBetween(min, max);  return energy; } //!< sets energy, randomised between min and max.  inline float setTemperature(float min, float max) {  temp = genFunc->uniformFloatBetween(min, max);  return temp; }//!< sets temperature, randomised between min and max.  inline float setOxygenRate(float min, float max) {  oxy = genFunc->uniformFloatBetween(min, max);  return oxy; } //!< sets oxygen, randomised between min and max.  inline int setEnvironmentID() {    return id; } //!< sets the environments id.  inline float setEnvironmentalCapacity(float energyAvail, float multiplier) {  cap = energyAvail \* multiplier;  return cap; } //!< sets environmental capacity, multiplies energy by multiplier.  std::shared\_ptr<GeneralFunctions> genFunc;//!< ref to General Functions class.  float energy; //!< holder for named var.  float temp; //!< holder for named var.  float oxy; //!< holder for named var.  float cap; //!< holder for named var.  float id; //!< holder for named var.  };  void EnvironmentCreation::environmentCreation(Environment& envir, float energyMin, float energyMax, float tempMin, float tempMax, float oxygenMin, float oxygenMax)  {  //set energy level.  envir.energyAvailable = setEnergyAvailable(energyMin, energyMax);  std::cout << "set environments energy level to: " << envir.energyAvailable << std::endl;  //set temperature level.  envir.temperature = setEnergyAvailable(tempMin, tempMax);  std::cout << "set environments temperature level to: " << envir.temperature << std::endl;  //set oxygen level.  envir.oxygenationRate = setEnergyAvailable(oxygenMin, oxygenMax);  std::cout << "set environments oxygen level to: " << envir.oxygenationRate << std::endl;  //set weight capacity.  envir.fEnvironmentCapacity = setEnvironmentalCapacity(envir.energyAvailable, envir.fCapacityMultiplier);  //need to set an environment ID too... maybe?  count++;  envir.ID = count;  std::cout << "set environments ID to: " << envir.ID << std::endl;  } |

*C01.01 – EnvironmentCreation .h and associated main function environmentCreation()*

|  |
| --- |
| struct Environment  {  float energyAvailable; //!< energy available in environment.  float temperature; //!< temperature of the environment.  float oxygenationRate; //!< oxygenation % of the environment.  float mutationModifier = 1.0f; //!< always initialised to 1 (no multiple effect), as initialisation considers environment 'clean', changes such a pollution will modify this.  float pollution = 0.0f; //!< at initalisation set to 0 as considered to be 'clean', pollution changes added later.  uint32\_t ID; //!< environmental ID number.  EnvironmentalStatus currentStatus; //!< the current status the environment is in.  float fEnvironmentCapacity = 0.0f; //!< capacity for combined population mass.  float fPopulationWeight = 0.0f; //!< combined weight of all the creatures.  float fCapacityMultiplier = 10.0f; //!< multiplier for getting environmentalCapacity, multiple this my energyAvailable.  float fWeightOffset = 100.0f; //!< offset to +/- population weight to environmental capacity, to give status' of SUSTAINABLE and PRESSURED.  //void updatePopulationWeight(AllSpecies allSpecies); //!< function to update the combined weight of the population in the environment.  inline float getEnergyAvailable() { return energyAvailable; }//!< get energyAvailable.  inline float getTemperature() { return temperature; } //!< get temperature.  inline float getOxygenRate() { return oxygenationRate; }//!< get oxygenation rate.  inline int getEnvironmentID() { return ID; } //!< get the environmental id.  }; |

*C01.02 – Struct Environment*

***C02: Creature Creation***

|  |
| --- |
| struct Creature  {  uint32\_t creatureID; //!< creature ID.  std::vector<float> geneStack;//!< vector containing the core gene values.    float thresholdScore; //!< score used to determine reproduction chances.  //energy vars.  float initialEnergyDemand; //!< initial energy demand of creature.  float finalEnergyDemand; //!< final energy demand of creature, with mutlipliers applying to this.  //oxygenation vars  float oxygenDemand; //!< oxygen demand value.  float oxygenRange; //!< range of oxygen to either side of the oxygen demand.  float oxygenTolMin; //!< oxygen toleration minimum, determined with oxygen demand and range.  float oxygenTolMax; //!< oxygen toleration maximum, determined with oxygen demand and range.  //ideal temperature vars.  float idealTemp; //!< ideal temperature value.  float idealTempRange; //!< ideal temperature range to either side of the ideal temperature.  float idealTempRangeMin; //!< ideal temperature minimum, determined with temperature demand and range.  float idealTempRangeMax; //!< ideal temperature maximum, determined with temperature demand and range.  //tolerated temperature vars.  float tolTempRange; //!< tolerated temperature range value, to be added on top of the ideal temperature values (creature can survive here but with a modifier to energy demands).  float tolTempRangeMin; //!< tolerated temperature minimum, determined with ideal temperatures and tolerated range.  float tolTempRangeMax; //!< tolerated temperature maximum, determined with ideal temperatures and tolerated range.  //bools for conditions.  bool isAlive; //!< determines whether creature is alive.  bool tempIdeal; //!< determines whether creature in ideal temperature range.  bool tempTol; //!< determines whether creature in tolerated temperature range.  bool oxyIdeal; //!< determines whether creature in ideal oxygen range.  bool oxyTol; //!< determines whether creature in tolerated oxygen range.  bool paired = false; //!< determines in reproduction stage whether creature has been paired up, initialised to false;  uint32\_t creatureNumber; //!< creation number within species.  CreatureSize creatureSize; //!< the size of the creature, determines life span, weight, litterSize; initialised to VERY\_SMALL.  float creatureWeight = 0.1f; //!< the 'weight' of an individual creature on the environment it is in, initialised to a genetic 0.1f.  float litterSize = 4.0f; //!< a float, if 5.45f, will definitely have 5 offspring and a 45% chance of 6.  int32\_t lifeSpan = 4; //!< if survives fitness tests, how many life spans the creature can survive. if 5.45, will definitely live 5 cycles, 45% chance to live 6.  }; |

*C02.01 – Struct Creature*

|  |
| --- |
| void CreatureCreation::creatureCreation(Creature & creature, float energyCentre, float energyGauss, float idealTempCentre, float idealTempGuass,  float idealTempRangeMin, float idealTempRangeMax, float tolTempRangeMin, float tolTempRangeMax, float oxyCentre, float oxyGauss,  float oxyRangeMin, float oxyRangeMax, float offspringMin, float offspringMax, uint32\_t lifeMin, uint32\_t lifeMax)  {  //pick creature size.  signed int randNum = genFunc->uniformIntBetween(0, 9);  if (randNum > 4)  randNum = 4;  creature.creatureSize = CreatureSize(randNum);  float fStartEnergy = 1.0f;  switch (creature.creatureSize)  {  case VERY\_SMALL:  //set creature weight, life span and litter size.  creature.creatureWeight = multiplier(creature.creatureWeight, 0.2f);  creature.lifeSpan = 1; //As very small, hard set to 1, as that is all a very small creature should live for.  creature.litterSize = multiplier(creature.litterSize, 2.0f);  fStartEnergy = multiplier(fStartEnergy, 100.0f); //CHANGE TO A RANDOM RANGE.  break;  case SMALL:  //set creature weight, life span and litter size.  creature.creatureWeight = multiplier(creature.creatureWeight, 0.6f);  creature.lifeSpan = multiplier(creature.lifeSpan, 0.5f);  creature.litterSize = multiplier(creature.litterSize, 1.5f);  fStartEnergy = multiplier(fStartEnergy, 200.0f); //CHANGE TO A RANDOM RANGE.  break;  case MEDIUM:  //set creature life span and litter size; leave weight the initialised standard.  plusMinusOne(creature.lifeSpan, creature.litterSize);  fStartEnergy = multiplier(fStartEnergy, 300.0f); //CHANGE TO A RANDOM RANGE.  break;  case LARGE:  //set creature weight, life span and litter size.  creature.creatureWeight = multiplier(creature.creatureWeight, 1.4f);  creature.lifeSpan = multiplier(creature.lifeSpan, 1.5f);;  creature.litterSize = multiplier(creature.litterSize, 0.5f);  fStartEnergy = multiplier(fStartEnergy, 400.0f); //CHANGE TO A RANDOM RANGE.  break;  case VERY\_LARGE:  //set creature weight, life span and litter size.  creature.creatureWeight = multiplier(creature.creatureWeight, 1.8f);  creature.lifeSpan = multiplier(creature.lifeSpan, 2.0f);  creature.litterSize = 1.0f; //As very large, hard set to 1, as that is all avery large creatures litter size should be.  fStartEnergy = multiplier(fStartEnergy, 500.0f); //CHANGE TO A RANDOM RANGE.  break;  }  creature.initialEnergyDemand = genFunc->normalFloatBetween(energyCentre, energyGauss);  //creature.initialEnergyDemand = genFunc->normalFloatBetween(fStartEnergy, energyGauss);    creature.initialEnergyDemand = genFunc->roundFloat(creature.initialEnergyDemand);  if (creature.initialEnergyDemand <= 0.0f)  { //CHOICE!!!  //do we remove the creature if their energy demand is less than 0...  //add to remove and delete list.  //or, do we randomly pick a constrained low number?  creature.initialEnergyDemand = resetVariable(1.0f, 20.0f, 60.0f, 80.0f);  }  fillGeneElement(creature, creature.geneStack, creature.initialEnergyDemand); //geneStack element (0) - initial energy demand  creature.idealTemp = genFunc->normalFloatBetween(idealTempCentre, idealTempGuass);  creature.idealTemp = genFunc->roundFloat(creature.idealTemp);  fillGeneElement(creature, creature.geneStack, creature.idealTemp); //geneStack element (1) - ideal temperature  creature.idealTempRange = genFunc->uniformFloatBetween(idealTempRangeMin, idealTempRangeMax);  fillGeneElement(creature, creature.geneStack, creature.idealTempRange); //geneStack element (2) - ideal temperature range  creature.tolTempRange = genFunc->uniformFloatBetween(tolTempRangeMin, tolTempRangeMax);  fillGeneElement(creature, creature.geneStack, creature.tolTempRange); //geneStack element (3) - tolerated temperature range.  creature.oxygenDemand = genFunc->normalFloatBetween(oxyCentre, oxyGauss);  creature.oxygenDemand = genFunc->roundFloat(creature.oxygenDemand);  if (creature.oxygenDemand <= 0.0f)  { //CHOICE!!!  //do we remove the creature if their oxygen demand is less than 0...  //add to remove and delete list.  //or, do we randomly pick a constrained low number?  creature.oxygenDemand = resetVariable(1.0f, 5.0f, 15.0f, 20.0f);  }  fillGeneElement(creature, creature.geneStack, creature.oxygenDemand); //geneStack element (4) - oxygen demand  creature.oxygenRange = genFunc->uniformFloatBetween(oxyRangeMin, oxyRangeMax);  fillGeneElement(creature, creature.geneStack, creature.oxygenRange); //geneStack element (5) - oxygen range  creature.idealTempRangeMax = creature.idealTemp + creature.idealTempRange;  creature.idealTempRangeMin = creature.idealTemp - creature.idealTempRange;  if (creature.idealTempRangeMin <= 0.0f)  creature.idealTempRangeMin = resetVariable(0.05f, 1.0f, 3.0f, 6.0f);  creature.tolTempRangeMax = creature.idealTempRangeMax + creature.tolTempRange;  creature.tolTempRangeMin = creature.idealTempRangeMin - creature.tolTempRange;  if (creature.tolTempRangeMin <= 0.0f)  creature.tolTempRangeMin = resetVariable(0.05f, 1.0f, 3.0f, 6.0f);  creature.oxygenTolMax = creature.oxygenDemand;  creature.oxygenTolMin = creature.oxygenDemand - creature.oxygenRange;  if (creature.oxygenTolMin <= 0.0f)  creature.oxygenTolMin = resetVariable(0.05f, 1.0f, 3.0f, 6.0f);  //add litterSize and creatureWeight to geneStack.  fillGeneElement(creature, creature.geneStack, creature.litterSize); //geneStack element (6) - litter size  fillGeneElement(creature, creature.geneStack, creature.creatureWeight); //geneStack element (7) - creature weight  //then add creatureSize and lifeSpan to geneStack.  fillGeneElement(creature, creature.geneStack, creature.creatureSize); //geneStack element (8) - creature size  fillGeneElement(creature, creature.geneStack, creature.lifeSpan); //geneStack element (9) - life span  creature.isAlive = true;  creature.tempIdeal = true;  creature.tempTol = true;  creature.oxyIdeal = true;  creature.oxyTol = true;  } |

*C02.02 – creatureCreation()*

|  |
| --- |
| class GeneralFunctions : public System  {  public:  virtual void start(SystemSignal init = SystemSignal::None, ...); //!< start the system.  virtual void stop(SystemSignal close = SystemSignal::None, ...); //!< stop the system.  static int32\_t uniformIntBetween(int32\_t min, int32\_t max); //!< get an int between min and max params.  static float uniformFloatBetween(float min, float max); //!< get a float between min and max params.  static int32\_t normalIntBetween(float c, float sigma); //!< get an int from a gaussian distribution described by C & sigma.  static float normalFloatBetween(float c, float sigma); //!< get an float from a gaussian distribution described by C & sigma.  static float roundFloat(float val); //!< func to round a float to 2 decimal floats.  static uint32\_t createNewCreatureID(uint32\_t id);  static uint32\_t createNewCreatureID(uint32\_t id, uint32\_t gen, uint32\_t child);  private:  static std::shared\_ptr<std::mt19937> s\_generator; //!< random number generating engine.  static std::uniform\_int\_distribution<int32\_t> s\_uniformInt; //!< uniform int distribution.  static std::uniform\_real\_distribution<float> s\_uniformFloat; //!< uniform float distribution.  static float s\_intRange; //!< range of int32.  static float s\_floatRange; //!< range of float.  }; |

*C02.03 – GeneralFunctions.h*

|  |
| --- |
| float CreatureCreation::resetVariable(float minLow, float minHigh, float maxLow, float maxHigh)  {  newMin = genFunc->uniformFloatBetween(minLow, minHigh);  newMax = genFunc->uniformFloatBetween(maxLow, maxHigh);  float result = genFunc->uniformFloatBetween(newMin, newMax);  return result;  } |

*C02.04 – resetVariable()*

***C03: Fitness Tests***

|  |
| --- |
| bool EnergyFitnessTest::fitnessTest(float creatRequired, float envirProvided)  {  bool result = true;  float demand = creatRequired;  float provided = envirProvided;  if (demand > provided)  result = false;  return result;  } |

*C03.01 – energy fitnessTest()*

|  |
| --- |
| bool TemperatureFitnessTest::fitnessTest(bool ideal, bool tolerated)  {  bool result = true;  bool inIdeal = ideal;  bool inTolerated = tolerated;  if (!inIdeal && !inTolerated)  result = false;  return result;  }  float TemperatureFitnessTest::multiplier(float energy, bool tolerated, bool ideal)  {  float result = energy;  if(!ideal && tolerated)  result = result \* 1.5f;  return result;  }  bool TemperatureFitnessTest::inRangeCheck(float max, float min, float env)  {  bool result = true;  float creatMax = max;  float creatMin = min;  float environ = env;  if (environ <= max  && environ >= min)  result = true;  else  result = false;  return result;  } |

*C03.02 – temperature fitnessTest() with multiplier() and inRangeCheck()*

|  |
| --- |
| bool OxygenFitnessTest::fitnessTest(float creatRequired, float envirProvided, bool tolerated)  {  bool result = true;  float demand = creatRequired;  float provided = envirProvided;  bool inTolerated = tolerated;  if (demand > provided && !inTolerated)  result = false;  return result;  }  float OxygenFitnessTest::multiplier(float energy, bool tolerated)  {  float result = energy;  if (tolerated)  result = result \* 1.5f;  return result;  }  bool OxygenFitnessTest::inRangeCheck(float max, float min, float env)  {  bool result = true;  float creatMax = max;  float creatMin = min;  float environ = env;  if (environ <= max  && environ >= min)  result = true;  else  result = false;  return result;  } |

*C03.03 – oxygenation fitnessTest() with multiplier() and inRangeCheck()*

|  |
| --- |
| void FullFitnessTest::creatureFitnessTests(CREAT &creature, Environment& environment)  {  //check that the creature is alive to do checks on in the first place.  if (creature.isAlive)  {  //test one - oxygenation fitness test.  creature.oxyIdeal = oxyFF.inRangeCheck(100.0f, creature.oxygenTolMax, environment.oxygenationRate);  creature.oxyTol = oxyFF.inRangeCheck(creature.oxygenTolMax, creature.oxygenTolMin, environment.oxygenationRate);  creature.isAlive = oxyFF.fitnessTest(creature.oxygenDemand, environment.oxygenationRate, creature.oxyTol);  if (creature.isAlive)  {  //test two - temperature fitness test.  creature.tempIdeal = tempFF.inRangeCheck(creature.idealTempRangeMax, creature.idealTempRangeMin, environment.temperature);  creature.tempTol = tempFF.inRangeCheck(creature.tolTempRangeMax, creature.tolTempRangeMin, environment.temperature);  creature.isAlive = tempFF.fitnessTest(creature.tempIdeal, creature.tempTol);  if (creature.isAlive)  {  //test three - energy fitness test.  creature.finalEnergyDemand = oxyFF.multiplier(creature.initialEnergyDemand, creature.oxyTol);  creature.finalEnergyDemand = tempFF.multiplier(creature.initialEnergyDemand, creature.tempTol, creature.tempIdeal);  creature.isAlive = energyFF.fitnessTest(creature.finalEnergyDemand, environment.energyAvailable);  }  }  //give the creature its threshold score.  if (creature.isAlive)  creature.thresholdScore = genFunc->roundFloat(environment.energyAvailable - creature.finalEnergyDemand);  }  } |

*C03.04 – combined full fitnessTest()*

|  |
| --- |
| void FullFitnessTest::creatureFitnessTests(CREAT & creature, Environment & environment, float energyReduction)  {  //take the energy reduction from the available energy in the environment.  //environment.energyAvailable = environment.energyAvailable - energyReduction;  float fCompeteAvailableEnergy = environment.energyAvailable - energyReduction;  //check that the creature is alive to do checks on in the first place.  if (creature.isAlive)  {  //test one - oxygenation fitness test.  creature.oxyIdeal = oxyFF.inRangeCheck(100.0f, creature.oxygenTolMax, environment.oxygenationRate);  creature.oxyTol = oxyFF.inRangeCheck(creature.oxygenTolMax, creature.oxygenTolMin, environment.oxygenationRate);  creature.isAlive = oxyFF.fitnessTest(creature.oxygenDemand, environment.oxygenationRate, creature.oxyTol);  if (creature.isAlive)  {  //test two - temperature fitness test.  creature.tempIdeal = tempFF.inRangeCheck(creature.idealTempRangeMax, creature.idealTempRangeMin, environment.temperature);  creature.tempTol = tempFF.inRangeCheck(creature.tolTempRangeMax, creature.tolTempRangeMin, environment.temperature);  creature.isAlive = tempFF.fitnessTest(creature.tempIdeal, creature.tempTol);  if (creature.isAlive)  {  //test three - energy fitness test.  creature.finalEnergyDemand = oxyFF.multiplier(creature.initialEnergyDemand, creature.oxyTol);  creature.finalEnergyDemand = tempFF.multiplier(creature.initialEnergyDemand, creature.tempTol, creature.tempIdeal);  creature.isAlive = energyFF.fitnessTest(creature.finalEnergyDemand, fCompeteAvailableEnergy); //change here, using the updated fTempEnergy.  }  }  //give the creature its threshold score.  if (creature.isAlive)  creature.thresholdScore = genFunc->roundFloat(environment.energyAvailable - creature.finalEnergyDemand);  }  } |

*C03.05 – updated fitness test to take competition causing energy reduction*

***C04: Population Creation***

|  |
| --- |
| uint32\_t GeneralFunctions::createNewCreatureID(uint32\_t id)  {  //convert ints to strings.  std::string s1 = std::to\_string(id);  //concatenate the strings.  std::string finalString = s1;  //convert concatenated string back into an int.  uint32\_t result = stoi(finalString);  //return the resulting int.  return result;  }  uint32\_t GeneralFunctions::createNewCreatureID(uint32\_t id, uint32\_t gen, uint32\_t child)  {  //convert ints to strings.  std::string s1 = std::to\_string(id);  std::string s2 = std::to\_string(gen);  std::string s3 = std::to\_string(child);  //concatenate the strings.  std::string finalString = s1 + s2 + s3;  //convert concatenated string back into an int.  int result = stoi(finalString);  //return the resulting int.  return result;  } |

*C04.01 – functions for creating creature IDs.*

***C05:Mutation Mechanic***

|  |
| --- |
| bool Mutation::mutationTest(float mutationChance)  {  float fURandomNumber = genFunc->uniformFloatBetween(0.0f, 1.0f); //float 0-1, uniform produces an even random spread within the range.  float fMutChance = mutationChance;  //mutation chance should be between 0-1, therefore a 5% chance should be 0.05f.  //this checks if it has been put in as 5% and reduces to the required value range.  if (fMutChance > 1.0f)  fMutChance = fMutChance \* 0.01f;  //this is a precautionary test, in chance user enters 0.5f wanting a 0.5% of mutation, this will result in a 50% chance of mutation.  else if (fMutChance <= 1.0f && fMutChance >= 0.25f)  std::cout << "WARNING - mutationChance is very high, between 25% and 100%; have you entered this correctly?" << std::endl;  if (fURandomNumber <= fMutChance)  return true;  else  return false;  }  void Mutation::mutationIntensity(float mutPercent, float &eleToMut, float envirMulti)  {  //get the value of the mutPercent of the elementToMutate so 10 intensity of element 5, gives a result of 0.5  float fMutPercentValue = (eleToMut \* 0.01f) \* mutPercent;  //get a random float using a gaussian function, the element to mutate being the centre and the sigma (range) being the mutation intensity.  float fMutElement = genFunc->normalFloatBetween(eleToMut, fMutPercentValue);  fMutElement = genFunc->roundFloat(fMutElement);  //TESTING, keeping count of positive and negative mutation to ensure both are happening and ratio between the two.  if (fMutElement > eleToMut) {  iPosMuts++;  fPosMutsTally += fMutElement;  }  else if (fMutElement < eleToMut) {  iNegMuts++;  fNegMutsTally += fMutElement;  }  //apply environmental multipliers to the mutation element.  fMutElement = fMutElement \* envirMulti;    eleToMut = genFunc->roundFloat(fMutElement);  } |

*C05.01 – mutationTest() & mutationIntensity()*

***C06: ‘Bacteria’ Stage Testing***

|  |
| --- |
| /ESPECIALLY USEFUL FOR COMPARING THE HIGHEST DIFFERENCE TO THE MUTATION INTENSITY.  //testing mutation chance & intensity functions.  float testRuns = 1000000;  int yesMut = 0;  int noMut = 0;  float mutChance = 2.5f;  float mutInten = 1.0f;  float enMulti = 1.0f;  if(mutInten > 0.05f)  {  //NOTE BELOW LINE ONLY REALLY WORKS IF mutInten is dividable by 0.05f.  int timesToRun = mutInten / 0.05f; //should be 20 if mutInten starts at 1.0f.  for (int i = 0; i < timesToRun; i++)  {  float diff = 0.0f;  float avDiff = 0.0f;  float highDiff = diff;  float largeMutList = 0;  float largeMutPercent = 0.0f;  for (int j = 0; j < testRuns; j++)  {  if (mut.mutationTest(mutChance) == true)  {  float element2Mutate = 100.0f;  mut.mutationIntensity(mutInten, element2Mutate, enMulti);  diff = element2Mutate - 100.0f;  //set to a positive value if negative.  if (diff < 0.0f)  diff = diff \* -1.0f;  //check if a large change.  if (diff >= 1.0f)  largeMutList = largeMutList + 1.0f;  //tally up the avDiff.  avDiff += diff;  //check if new diff is larger than highDiff, if so, set highDiff to the diff value.  if (diff > highDiff)  highDiff = diff;  //std::cout << "new mutated element value is: " << element2Mutate << std::endl;  yesMut++;  }  else  noMut++;  }  std::cout << "MUTATION INTENSITY IS: " << mutInten << std::endl;  std::cout << "highest difference is: " << highDiff << std::endl;  avDiff = avDiff / testRuns;  std::cout << "average difference is: " << avDiff << std::endl;  largeMutPercent = largeMutList / testRuns;  std::wcout << "percentage of mutations over 1% change is: " << largeMutPercent << std::endl << std::endl;  mutInten = mutInten - 0.05f;  }  //std::cout << "Over " << testRuns << " tests, number of YES mutations is: " << yesMut << " and number of NO mutations is: " << noMut << std::endl;  } |

*C06.01 – Mutation Testing Setup*

***C07: Selection Mechanic***

|  |
| --- |
| class Selection  {  public:    Selection() {}; //!< constructor.  ~Selection() {}; //!< deconstructor.  std::vector<std::pair<float, std::vector<float>>> parentSelection(SpeciesInfo species);  void fillSelectionTable(SpeciesInfo species);  void fillReproductionTable(std::vector<std::pair<float, std::vector<float>>> selectTable, uint32\_t numToReprod);  uint32\_t getSelectionPercent(std::vector<std::pair<float, std::vector<float>>> selectTable);  inline void setSelectionPercentage(uint32\_t desiredPercentage) { selectionPercentage = desiredPercentage; };  float selectionPercentage = 30.0f;  private:  Crossover cross; //!< ref to Crossover class.  std::vector<std::pair<float, std::vector<float>>> selectionTable; //!< vector of pairs, contains a float (for the threshold score) and another vector of floats (for the gene stack of the creature).  std::vector<std::pair<float, std::vector<float>>> toReproduceTable; //!< vector of floats vectors, containing the gene stack information for those creatures which will reproduce.    uint32\_t percentToReproduce;  }; |

*C07.01 – Selection.h*

|  |
| --- |
| std::vector<std::pair<float, std::vector<float>>> Selection::parentSelection(SpeciesInfo species)  {  //clear everything ready for use.  selectionTable.clear();  toReproduceTable.clear();  percentToReproduce = 0;  //fill the selection table with paired scores and geneStacks of the creatures from the passed in species.  fillSelectionTable(species);  //sort the table in descending order using the first of pair, ie the scores.  std::sort(selectionTable.begin(), selectionTable.end(), greater());  //take the n% to be used for reproduction.  percentToReproduce = getSelectionPercent(selectionTable);  //populate reproduction table with selected from selection table.  fillReproductionTable(selectionTable, percentToReproduce);  //return the table to be reproduced.  return toReproduceTable;  } |

*C07.02 – parentSelection()*

|  |
| --- |
| void Selection::fillSelectionTable(SpeciesInfo species)  {  //fail safe, if filling with a new species data, the selectionTable should be clear, check that...  if (selectionTable.size() == 0)  {  //get the size of species membership.  uint32\_t size = species.speciesMembership.size();  //reserve this space in the selection table, more efficient than increasing size one element at a time in the below loop.  selectionTable.reserve(size);  //fill the selection table with the paired variables of the creatures threshold score and their gene stacks.  for (int i = 0; i < size; i++)  selectionTable.push\_back(std::make\_pair(species.speciesMembership.at(i).thresholdScore, species.speciesMembership.at(i).geneStack));  }  else  std::cout << "WARNING - SELECTION - fillSelectionTable() - trying to add two different species to same selectionTable" << std::endl;  } |

*C07.03 – fillSelectionTable()*

***C08: Crossover Mechanic***

|  |
| --- |
| std::vector<Creature> Crossover::fullCrossover(std::vector<std::pair<float, std::vector<float>>> parents)  {  //clear the temporary gene stack and new creatures vector ready for use.  tempNewGeneStack.clear();  tempNewCreatures.clear();  //get the relevant gene stacks from parents.  getGeneStacks(parents);  //pair these parents and their gene stacks up.  pairParents(geneStacksToPairVec);  //loop through paired parents.  for(int i = 0; i < pairedParents.size(); i++)  {  //determine and get the number of offspring this pair of parents will have.  uint32\_t tempOffspringNum = getOffspringNumber(pairedParents.at(i));  //loop through and repeat creature creation stage for the number of offspring.  for (int j = 0; j < tempOffspringNum; j++)  {  //a new gene stack from the paired parents.  crossoverGeneStacks(pairedParents.at(i), pairedParents.at(i).first.size());  //use new gene stack to create a new offspring creature and add to the temporary new creatures vector.  tempNewCreatures.push\_back(cc.createCreatureFromGeneStack(tempNewGeneStack));  }  }  //return the creature.  return tempNewCreatures;  } |

*C08.01 – fullCrossover()*

|  |
| --- |
| void Crossover::pairParents(std::vector<std::pair<std::vector<float>, bool>>& parents)  {  //clear pairedParents ready for use.  pairedParents.clear();  //NOTE - how to pair parents?  //iterate down through the list and match that way?  //or randomly select them from the list? May take too long randomly picking numbers until it gets the correct last one.  for (int i = 0; i < parents.size(); i++)  {  //check that the creature hasn't already been assigned, if their second pair is false.  if (parents.at(i).second == false)  {  //pair creature at i and i+1  pairedParents.push\_back(std::make\_pair(parents.at(i).first, parents.at(i + 1).first));  //set the bool to true for both of these parents, as saying that they are assigned.  parents.at(i).second = true;  parents.at(i + 1).second = true;  //i++ as skipping a creature as it is assigned.  i++;  }  else  std::cout << "WARNING - CROSSOVER - pairParents() - creature already paired up" << std::endl;  }  } |

*C08.02 – pairParents()*

***C09: Species Mechanic***

|  |
| --- |
| struct SpeciesInfo  {  uint32\_t speciesID = 0; //!< ID of species, taken from the seed population member ID, initialised to 0 as an easy to tell to see if it has been assigned, ie. 0 means it hasn't been assigned.  std::vector<Creature> speciesMembership; //!< vector of all creatures that are members of this species.  uint32\_t currentMembers; //!< number of current members that are alive in the species.  uint32\_t previousMembers; //!< number of last cycles membership.  uint32\_t startCycleMembers; //!< membership at start of cycle.  uint32\_t endCycleMembers; //!< membership at end of cycle.  uint32\_t cycleOffspringCount;//!< count of offspring this cycle.  uint32\_t cycleTotalDeadCount;//!< count of total dead this cycle.  uint32\_t cycleFailedFitnessDeadCount;//!< count of dead due to failed fitness test this cycle.  uint32\_t cycleOldAgeDeadCount;//!< count of dead due to old age this cycle.  uint32\_t test = 0;  std::vector<float> seedGeneStack; //!< the gene stack of the seed members of this species.  std::vector<float> speciesGeneStack;//!< the current averaged gene stack of all currently alive members of the species.    bool speciesAlive = true; /!< bool to determine whether species still has members, initalised to true;    void speciesAliveCheck() {  if (currentMembers == 0)  speciesAlive = false;  else  speciesAlive = true;  }//!< check whether the species has current members, so still going.  void addValueToUIVar(uint32\_t& variableToAlter, uint32\_t byValue) {  variableToAlter += variableToAlter + byValue;  } //!<  }; |

*C09.01 – struct SpeciesInfo*

|  |
| --- |
| struct AllSpecies  {  //std::vector<SpeciesInfo> fullSpeciesVec; //!< a full list of all species.  std::vector<SpeciesInfo> aliveSpeciesVec; //!< all alive species.  std::vector<SpeciesInfo> extinctSpeciesVec; //!< all extinct species.  uint32\_t totalSpeciesCount; //!< running count of all species.  uint32\_t aliveSpeciesCount; //!< count of all alive species.  uint32\_t extinctSpeciesCount; //!< count of all now dead species.  }; |

*C09.02 – struct AllSpecies*

|  |
| --- |
| lass Species  {  public:  Species(); //!< default constructor.  ~Species() {}; //!< deconstructor.  void createNewSpecies(SpeciesInfo& species, std::vector<float>& geneStack, uint32\_t creatureID); //!< create a new species, adding the relevant creatures to this species vector    void assignSpeciesToAllSpeciesVector(SpeciesInfo species, std::vector<SpeciesInfo>& speciesVector, AllSpecies& allSpecies); //!< assign a species to the species vector.  void addCreatureToSpecies(Creature creature, SpeciesInfo& species); //!< add a creature to a species.  void checkSpeciesDivergence();//\*\*\*\*TO-DO\*\*\*\* //!< check membership gene stacks against species gene stack to see if creature has diverged.  void startCycleMemberships(AllSpecies &allSpec);  void endCycleMemberships(AllSpecies &allSpec);  void resetCycleCounts(AllSpecies &allSpec);  void updateSpeciesMembershipCounts(SpeciesInfo& species); //!< update species membership, adding new members and removing 'dead' ones.  std::vector<float> getSeedGeneStack(SpeciesInfo species); //!< get the species initial seed population gene stack.  std::vector<float> getSpeciesGeneStack(SpeciesInfo species); //!< get the species current populations gene stack.    void updateSpeciesGeneStack(SpeciesInfo& species); //!< get the average of the current species population gene stack and update/set the species current gene stack.  void updateAllSpecies(AllSpecies& allSpecies); //!< update the species vectors with full, alive and extinct.  int32\_t setVariable(int32\_t valToSetToo) { int32\_t result = valToSetToo; return result; } //!<  private:  void updateAllSpeciesCounts(AllSpecies& allSpecies); //!< updates the counts of the species vectors with the size of the vectors.  SpeciesInfo si; //!< reference to SpeciesInfo struct.  AllSpecies as; //!< reference to AllSpecies strcut.  std::shared\_ptr<GeneralFunctions> genFunc;//!< pointer ref to general functions class.  }; |

*C09.03 – class Species*

|  |
| --- |
| std::vector<float> Species::getSpeciesGeneStack(SpeciesInfo species)  {  std::vector<float> result;  result.reserve(species.speciesGeneStack.size());  result.assign(species.speciesGeneStack.begin(), species.speciesGeneStack.end());  return result;  }  //void Species::updateSpeciesGeneStack(std::vector<Creature> speciesMembership, std::vector<float> & speciesGeneStack)  void Species::updateSpeciesGeneStack(SpeciesInfo & species)  {  //ensure speciesGeneStack has values  if (species.speciesGeneStack.size() == 0)  species.speciesGeneStack.resize(species.speciesMembership.at(0).geneStack.size());  //iterate through membership of species for each gene element of the their species gene stack.  for (int i = 0; i < species.speciesMembership.size(); i++)  {  //iterate through gene stack of species member.  for (int j = 0; j < species.speciesMembership.at(i).geneStack.size(); j++)  {  //add the gene stack values of each member to the species overall average gene stack.  species.speciesGeneStack.at(j) = species.speciesGeneStack.at(j) + species.speciesMembership.at(i).geneStack.at(j);  }  }  //now average each element of the species gene stack by the size of the species membership.  for (int i = 0; i < species.speciesGeneStack.size(); i++)  {  float result = species.speciesGeneStack.at(i);  result = genFunc->roundFloat(result / species.speciesMembership.size());  species.speciesGeneStack.at(i) = result;  }  } |

*C09.04 – gene stack extensions to species class.*

***C10: Gene Stack Mechanic***

|  |
| --- |
| Creature CreatureCreation::createCreatureFromGeneStack(std::vector<float> newGeneStack)  {  //create a new creature object.  Creature newCreature;  //reserve its geneStack size to the newGeneStackSize, more efficient than resizing element by element.  newCreature.geneStack.reserve(newGeneStack.size());  //loop through and set creatures gene stack to the new gene stack.  for (int i = 0; i < newGeneStack.size(); i++)  newCreature.geneStack.push\_back(newGeneStack.at(i));  //create the rest of the creatures variables from this genestack.  updateCreature(newCreature);  //REMEMBER - in main run ft.creatureFitnessTests on these creatures to complete build, specifically threshold score and final energy demand, VERY IMPORTANT.  //return the newly created creature.  return newCreature;  } |

*C10.01 – createCreatureFromGeneStack()*

***C11: Environmental Capacity Mechanic***

|  |
| --- |
| if(allSpecies.aliveSpeciesVec.size() != 0)  {  //iterate through per species.  for (int i = 0; i < allSpecies.aliveSpeciesVec.size(); i++)  {  float fSpeciesWeight = allSpecies.aliveSpeciesVec.at(i).speciesGeneStack.at(7); //element 7 of gene stack is species creature weight.  int iSpeciesSize = allSpecies.aliveSpeciesVec.at(i).speciesMembership.size();  float fTotalSpeciesWeight = iSpeciesSize \* fSpeciesWeight;  //add to the total combined population weight.  envir[0].fPopulationWeight += fTotalSpeciesWeight;  }  } |

*C11.01 – tally total creature weight in an environment.*

***C12: Creature Size Mechanic***

|  |
| --- |
| enum CreatureSize { VERY\_SMALL, SMALL, MEDIUM, LARGE, VERY\_LARGE }; |

*C12.01 – enum CreatureSize*

***C13: Creature Competition Mechanic***

|  |
| --- |
| enum EnvironmentalStatus { ABUNDANCE, SUSTAINABLE, PRESSURED, CRITICAL, FAMINE }; |

*C13.01 – enum EnvironmentalStatus*

|  |
| --- |
| void Competition::setEnvironmentalStatus(Environment &environment)  {  int temp = SUSTAINABLE;  //compare population weight to environmental capacity and set environmental status accordingly.  if (environment.fPopulationWeight <= (environment.fEnvironmentCapacity \* fAbundanceMultiplier)) //0.0f to 0.75f  {  //set environment status to ABUNDANCE...  temp = ABUNDANCE;  environment.currentStatus = EnvironmentalStatus(temp);  }  else if (environment.fPopulationWeight > (environment.fEnvironmentCapacity \* fAbundanceMultiplier) && //0.75f  environment.fPopulationWeight <= environment.fEnvironmentCapacity) //0.75f to 1.0f  {  //set environment status to SUSTAINABLE...  temp = SUSTAINABLE;  environment.currentStatus = EnvironmentalStatus(temp);  }  else if (environment.fPopulationWeight > environment.fEnvironmentCapacity &&  environment.fPopulationWeight <= (environment.fEnvironmentCapacity \* fPressuredMultiplier)) //1.0f to 1.25f  {  //set environment status to PRESSURED...  temp = PRESSURED;  environment.currentStatus = EnvironmentalStatus(temp);  }  else if (environment.fPopulationWeight > (environment.fEnvironmentCapacity \* fPressuredMultiplier) &&  environment.fPopulationWeight <= (environment.fEnvironmentCapacity \* fCriticalMultiplier)) //1.25f to 1.5f  {  //set environment status to CRITICAL...  temp = CRITICAL;  environment.currentStatus = EnvironmentalStatus(temp);  }  else if (environment.fPopulationWeight > (environment.fEnvironmentCapacity \* fPressuredMultiplier)) //1.5f>>>  {  //set environment status to FAMINE...  temp = FAMINE;  environment.currentStatus = EnvironmentalStatus(temp);  }  else  {  //else default to sustainable.  temp = SUSTAINABLE;  environment.currentStatus = EnvironmentalStatus(temp);  }  }  void Competition::getEnvironmentStatus(Environment &environment)  {  fToCompetePercent = 1.0f;  switch (environment.currentStatus)  {  case ABUNDANCE:  //abundance == more resources available, but what is the effect? A reverse of the below with added energy? More offspring?  fToCompetePercent = 0.0f;  break;  case SUSTAINABLE:  //sustainable == no extra effects, good or bad.  fToCompetePercent = 0.0f;  break;  case PRESSURED:  //pressured == 30% of population experiences a reduction of available energy, as competing for food.  fToCompetePercent \*= fPressuredState;  break;  case CRITICAL:  //critical == 60% of population experiences a reduction of available energy, as competing for food.  fToCompetePercent \*= fCriticalState;  break;  case FAMINE:  //famine == 100% of population experiences a reduction in available energy, as competing for food.  fToCompetePercent \*= fFamineState;  break;  }  } |

*C13.02 – get & set environment status functions.*

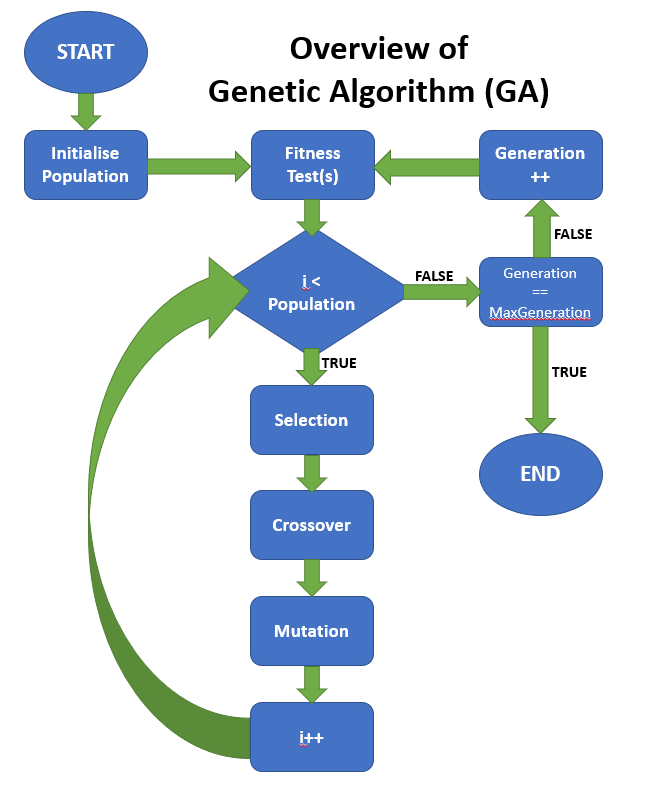
|  |
| --- |
| class Competition  {  public:  void setEnvironmentalStatus(Environment &environment);  void getEnvironmentStatus(Environment &environment);  float getPopulationToCompete(int speciesMembershipSize, int sizeGeneElement);  private:  float fToCompetePercent = 1.0f;  int iCreatureSIZE = 0;  int iFinalPopToCompete = 0;  float fCriticalMultiplier = 1.5f; //!< multiplier for environment capacity/creature weight comparisons.  float fPressuredMultiplier = 1.25f; //!< multiplier for environment capacity/creature weight comparisons.  float fAbundanceMultiplier = 0.75f; //!< multiplier for environment capacity/creature weight comparisons.  float fVSmall = 0.95f; //!< % of very small creature size populations to have competition mechanics applied to.  float fSmall = 0.90f; //!< % of small creature size populations to have competition mechanics applied to.  float fMedium = 0.80f; //!< % of medium creature size populations to have competition mechanics applied to.  float fLarge = 0.70f; //!< % of large creature size populations to have competition mechanics applied to.  float fVLarge = 0.60f; //!< % of very large creature size populations to have competition mechanics applied to.  float fPressuredState = 0.3f; //!< % of overall populations to have competition mechanics applied to if environmental status is PRESSURED.  float fCriticalState = 0.6f; //!< % of overall populations to have competition mechanics applied to if environmental status is CRITICAL.  float fFamineState = 0.99f; //!< % of overall populations to have competition mechanics applied to if environmental status is FAMINE.  }; |

*C13.03 – creature size multipliers in competition.h*

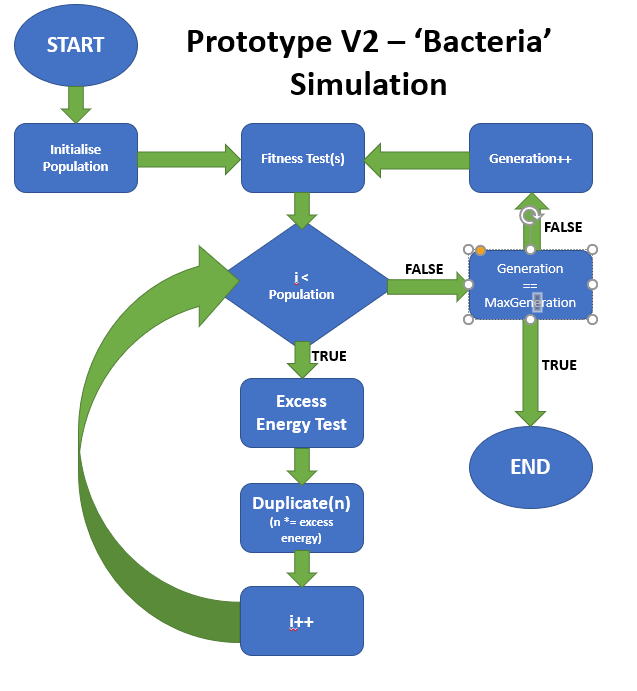
|  |
| --- |
| //FOR TESTING...  envir[0].fPopulationWeight = 500.0f; //for ABUNDANCE  envir[0].fPopulationWeight = 900.0f; //for SUSTAINABLE  envir[0].fPopulationWeight = 1100.0f; //for PRESSURED  envir[0].fPopulationWeight = 1400.0f; //for CRITICAL  envir[0].fPopulationWeight = 1700.0f; //for FAMINE |

*C13.04 – testing using hard set values.*

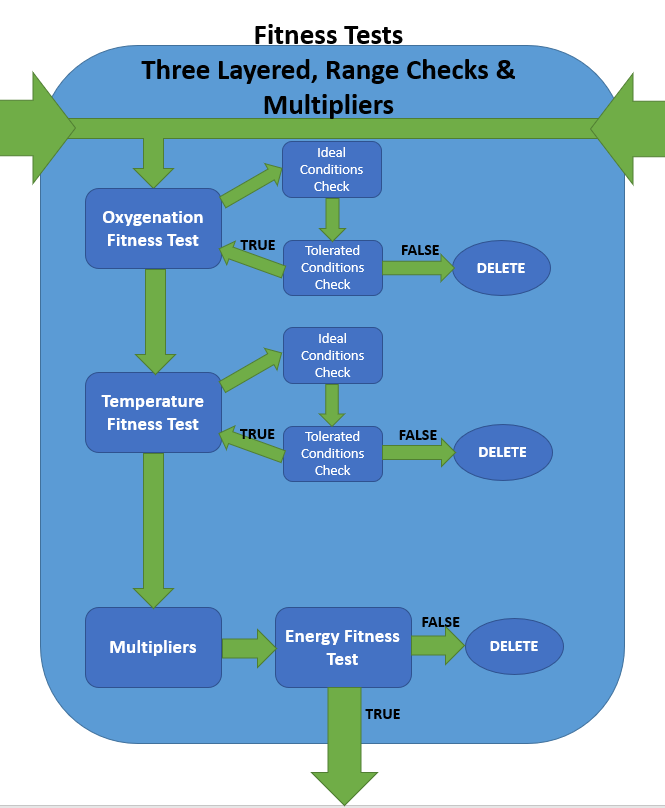
***DESIGNS***

****

*D01.01 – Overview of Genetic Algorithm*

****

*D01.02 – Prototype ‘Bacteria’ Stage Design*

****

*D01.03 – Fitness Tests Design*

***DOCUMENTATION***

***E01: Ethical Review***

1.Student Name: Tom Hodgson

2.P-number: P2420642

3.Programme: BSc Computer Games Programming

4.Email address: [p2420642@my365.dmu.ac.uk](mailto:p2420642@my365.dmu.ac.uk)

5.Project Title: Evolutional Simulated Model

6.Supervisor: Jethro Shell

7.Brief description of proposed activity and its objectives:

To create a simulated model of evolution using the theories of evolutional and genetic algorithms. Creating a varied, changeable environment and starting with a population of a single ‘creature’, aim to populate that environment with a diverse range of evolved ‘creatures’.

8.Ethical Issues Identified:

No ethical issues have been identified.

9.How these will be addressed:

None identified.

10.Has the project proposal identified any of the following research procedures?

|  |  |
| --- | --- |
| 1. Gathering information about human beings through: Interviewing, Surveying, Questionnaires, Observation of human behaviour | NO |
| 2. Using archived data in which individuals are identifiable | NO |
| 3. Researching into illegal activities, activities at the margins of the law or activities that have a risk of personal injury | NO |
| 4. Supporting innovation that might impact on human behaviour e.g. Behavioural Studies | NO |

11.If ‘Yes’ to any of 1-4 above: have you considered the following?

|  |  |
| --- | --- |
| 1. Providing participants with full details of the objectives of the research |  |
| 2. Providing information appropriate for those whose first language is not English |  |
| 3. Researching into illegal activities, activities at the margins of the law or activities that have a risk of personal injury |  |
| 4. Voluntary participation with informed consent |  |
| 5. Written description of involvement |  |
| 6. Freedom to withdraw |  |
| 7. Keeping appropriate records |  |
| 8. Signed acknowledgement and understanding by participants |  |
| 9. Consideration of relevant codes of conduct/guidelines |  |

12.Ethical Review Outcome: *If the outcome is no. 3 or 4, this form should be forwarded to the Faculty Research Ethics Committee.*

1. **No ethical issues**

2. Minor ethical issues which have been addressed and concerns resolved

3. Major ethical issues which have been addressed and concerns resolved

4. Ethical issues that have not been resolved/addressed

13.Student Signature: Tom Hodgson

14.Supervisor Signature:

15.Date: 05/11/2020

***E02: Global Checklist***

1.Student Name: Tom Hodgson

2.P-number: P2420642

3.Programme: BSc Computer Games Programming

4.Email address: [p2420642@my365.dmu.ac.uk](mailto:p2420642@my365.dmu.ac.uk)

5.Project Title: Evolutional Simulated Model

6.Supervisor: Jethro Shell

7.Please indicate which of these possible attributes is addressed by your undertaking of this project. You should select at least two items.

|  | Addressed by Project? |
| --- | --- |
| 1- Ability to work collaboratively: teams from a range of backgrounds and countries |  |
| 2- Excellent communication skills with a sensitivity to speaking with and listening to non-native English speakers |  |
| 3- An ability to embrace multiple perspectives and challenge thinking in a range of cultural context |  |
| 4- A capacity to develop new skills and behaviours according to role requirements | YES |
| 5- An ability to negotiate and influence clients across the globe from different cultures |  |
| 6- An ability to form professional, global networks |  |
| 7- An openness to/respect of a range of perspectives from around the world | YES |
| 8- Multi-cultural learning agility (i.e. able to learn in any culture or environment) |  |

8.Brief description of how the ticked attributes have been addressed (max. 50 words per item):

No.4 – Evolutional and genetic algorithms, along with their programming are completely new to me; understanding and implementing them will be a new skill.

No.7 – Many researchers in this field are from all around the world, particularly the far east. Cultural influences do effect research and I am open and respectful to all.

9.Student Signature: Tom Hodgson

10.Supervisor Signature:

11.Date: 05/11/2020

***E03: Project Contract***

1.Student Name: Tom Hodgson

2.P-number: P2420642

3.Programme: BSc Computer Games Programming

4.Email address: [p2420642@my365.dmu.ac.uk](mailto:p2420642@my365.dmu.ac.uk)

5.Project Title: Evolutional Simulated Model

6.Project Proposer: Tom Hodgson

7.Supervisor: Jethro Shell

8.Introduction (max. 100 words):

* For my final year project, I have chosen to take on a task which will be challenging and hopefully, very rewarding. The aim is to create a project that I am passionate about and showcases what I have learnt throughout my degree.
* I plan for this project to be a piece of work that will be relevant within the games industry, a piece of work to discuss in interviews; whilst at the same time academically challenging.

9.Project Background (max. 300 words):

* The background for this project is inspired by the ‘God’ genre of gaming in which a world is created via simulated models and the user then interacts with that world by changing some of the stats of the simulation in order to produce a change within that world.
* This simulated world is of great interest; the potential intelligence, sophistication and realism would be ideal to create a basis for an entertaining, challenging and varied piece of entertainment which a high degree of replayability.

10.Aims (max. 100 words):

* Create a varied environment in which a modelled simulation of evolution would take place.
* Starting with a small population of a single ‘creature’, over numerous iterations, developing into numerous separate ‘creatures’ optimally suited to the various differing conditions within the environment.
* Represent this via a basic graphical rendering of the location and density of the various creatures.
* Extensive use of the principles of evolutionary and genetic algorithms.
* To be achieved using C++ within visual studios.

11.Objectives (max. 200 words):

* Create a system for an **environment**, which will influence the evolution of the ‘creatures’.
* Allow the **user to alter** certain settings of the **environment**, which in turn will have a different influence on the evolution of the ‘creatures’.
* Create a system which **stores and uses a number of variables** for each creature which in turn will be used to determine whether this creature has survived and whether it has reproduced.
* Create a system for **survival**, in which it is determined whether the creature has successfully been able to survive.
* Create a system for **reproduction**, in which it is determined which ‘creatures’ will be considered as successfully reproducing and passing on their genetic code.
* Create a system for combination of parental genetic material (**crossover**) and then the inclusion of new genetic material (**mutation**) to create the next generation of ‘creatures’.
* Create a system that takes into account **internal** and/or **external** **features** that will have an impact on the evolution of ‘creatures’.
* Create a **basic UI system**.
* Render the above information in a **basic graphical representation**.

12.Deliverables (max. 100 words):

* The main system, a piece of software that will run the project as described above.
* All of the code that has been written for the software.
* A full report explaining the project, describing its component, the development cycle and a critical analysis of the process.
* A viva presentation explaining the project and its development process, followed by a Q&A.
* Documents for at least ten meeting with the project supervisor.

13.Resources and Constraints (max. 100 words):

* Hardware:
  + Laptop, monitors, external hard drive.
* Software:
  + Visual studios, notepad++
* Libraries:
  + OpenGL, C standard
  + Potentially: SFML, Box2D (TBC)
* De Montfort University:
  + Academics within the CEM department.

14.Sources of Information (max. 100 words):

* De Montfort University library resources.
* Packt Publishers extensive content.
* Academic websites including:
  + [www.ieee.org](http://www.ieee.org)
  + [www.omicsonline.org](http://www.omicsonline.org)
  + [www.scholar.google.com](http://www.scholar.google.com)
  + [www.royalsociety.org](http://www.royalsociety.org)
  + [www.core.ac.uk](http://www.core.ac.uk)
* Academic Books, Journals & Articles:
  + Effective Prediction with Machine Learning by Julian Avila.
  + Hands-On Genetic Algorithms with Python by Eyal Wirsansky.
  + An Improved Genetic Algorithm by Ping Ma, Meng Tian & Ming Yang
* Academics:
  + Professor Ziheng Yang
  + Antonios Liapis

15. Risk Analysis (max. 100 words):

* Full or partial loss of software data.
  + Weekly backup on an external hard drive.
  + Regular pushes to a github repository.

16.Schedule of Activities (max. 300 words):

* Research
  + Research into the theory and then uses of evolutional and genetic algorithms.
  + Research for similar uses to the project of these algorithms.
  + Design and plan the software.
    - By the 24th of December.
* Basic UI: code and implement a basic UI system for the software
  + - By the 24th of December.
* *First Deliverable – 15th January 2021.*
* Environment – create the basis of the environment, with its various variables.
  + - By the end of January.
* 1st Step – basic ‘bacteria’ creature.
  + This basic creature will be a stripped back version of the planned more complex water creature. They will have a shorter lifespan and higher reproduction rate for accelerated evolution.
  + Basic survival system – with fewer variable taken into account.
    - By the middle of February.
  + Basic reproduction system - asexual reproduction, so will only feature the mutation stage of the algorithm and be without the combination stage of the parents genetic material.
    - By the end of February
* 2nd Step – More complex ‘water-based’ creatures.
  + This creature will be far more complex with two parent reproduction and variations in diets, size, lifespan and behaviour.
  + Advanced survival system – greater number of variables for survival and differing strategies for survival.
    - By the end of March.
  + Advanced reproduction system – full two parent sexual reproduction which will now include the combination stage of the parents genetic material.
    - By the middle of March.
* User input and alteration of environment.
  + - Middle of April
* Basic graphical representation.
  + A basic representation of location and density of various creatures in world.
    - Middle of April
* Predator/Prey system mechanic.
  + Implement of system for judging success/failure of predators.
    - End of April
* Main report.
  + Complete the main report for the project.
* *Final Deliverable – 7th of May 2021.*

17.Student Signature:

Tom Hodgson

18.Supervisor Signature:

19.Date:

05/11/2020

1. *Gunturu, M., Shakarad, G. N., and Singh, (2017) S. Fitness function to find game equilibria using genetic algorithms, 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1531-1534.* [↑](#footnote-ref-1)
2. *Martínez-Arellano G., Cant R. & Woods D., (2017) Creating AI Characters for Fighting Games Using Genetic Programming, in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 4, pp. 423-434, Dec. 2017.* [↑](#footnote-ref-2)
3. *Koza J. R., (1999) Darwinian invention and problem solving by means of genetic programming, IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), Tokyo, Japan, 1999, pp. 604-609 vol.3.* [↑](#footnote-ref-3)
4. *Kholimi A. S., Hamdani A. & Husniah L., (2018) Automatic Game World Generation for Platformer Games Using Genetic Algorithm, 2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Malang, Indonesia, 2018, pp. 495-498.* [↑](#footnote-ref-4)
5. *Cole N., Louis S. J. & Miles C., (2004) Using a genetic algorithm to tune first-person shooter bots, Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 139-145 Vol.1.* [↑](#footnote-ref-5)
6. *Martínez-Arellano G., Cant R. & Woods D., (2017) Creating AI Characters for Fighting Games Using Genetic Programming, in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 4, pp. 423-434, Dec. 2017* [↑](#footnote-ref-6)
7. *Rawat, S. S., Rawat, K. S., Rawat, V., & Nijhawan, R. (2020) Neural Networks based Hand-crafted genetic learning approach to simulate Space Mario Game, 2020 International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2020, pp. 1-5.* [↑](#footnote-ref-7)