Genetic Fuzzy Based Controller for Space Invaders Player

Sam King

Department of Aerospace Engineering, University of Cincinnati

king3sl@ucmail.uc.edu

**Abstract:** Space Invaders is an arcade game, released in 1978, and is considered one of the most influential games of all time. It is also an effective simulation environment for different control schemes. The objective of the game is to control a player and defeat wave after wave of enemies that are streaming, vertically, down the screen. Traditionally the player can move laterally and vertically. For this paper, the player is constrained to one position but is free to rotate around its center. The objective of this project is to create a Genetic Fuzzy System (GFS), that is trained by the game itself, to systematically eliminate enemies, generated at random. The system is trained to combat multiple enemies, generated at random across the entire width of the game arena. Two training methods were used for the system. Here I detail the methodology and structure for the system and compare the results of the different training methods. I then applied these systems to play the game and compared the results. Both methods produced favorable results and the results are presented here. Also, these methods produced players that were effective in playing the game Space Invaders.

# Introduction:

## Problem

Create a Cascading Genetic Fuzzy System, trained by a Genetic Algorithm, that can play the game Space Invaders and is effective against multiple enemies.

## Main Purpose

Create a system, along with an architecture, that can later be adapted and scaled for real world scenarios. Also, compare training methods to determine which method produces the best results and use these insights to improve the system in the future.

## Approach

The simplest form of controller for this scenario would be a single Fuzzy Inference System (FIS) with multiple inputs and multiple outputs that is trained with a Genetic Algorithm (GA). That system, however, would be very computationally expensive and slow to respond in certain scenarios. Instead, I designed a system that includes three cascading FIS’s, trained by GA, an enemy sorting function to create a system that is effective and computationally efficient in training and operation. By transforming the system into a cascading fuzzy style, I was able to keep the chromosome length at a reasonable level and keep the rules necessary for operation to a minimum. This also helped maximize diversity amongst the populations created by GA.

The system was constructed and trained with two different training methods. The first method trains the system by competing against a single enemy with one laser available repeated 5 times. The second system is trained by competing against ever increasing waves of enemies. These methods will be detailed later.

## Nature of the Case Study

The game Space Invaders was chosen because it is a well-known game, and a simplistic simulation environment for testing control schemes. On top of that, the game can be programmed in Python and configured to easily interface with the controller. The schematic of system variables can be seen in *Figure 1*.

The Player has 1 degree of freedom, rotation, that is measured from the vertical by the variable α. Each enemy also has 1 degree of freedom. The enemies are generated at random across the width of the game area and move in the positive Y direction at a constant rate of speed. The number of enemies starts with 2 and increases by 2 after each subsequent level.

The other variables are a steering command, provided by the steering FIS, as well as an aiming point, generated by the aiming FIS. The outputs of these two FIS’s are then fed into the firing control FIS that tells the player when to fire. This is the extent of the game for this paper and constitutes a 3 input 2 output system that is applied to *n* enemies simultaneously.

## Why Genetic Fuzzy?

A Fuzzy Inference System (FIS) is an explainable artificial intelligence method. A FIS uses fuzzification, rule-inference, and defuzzification to make decisions. Genetic Algorithm is an evolutionary computing method designed to operate as a multi-criteria optimization method. A FIS is a powerful tool because it can draw on an expert knowledge base, but it lacks strong learning capabilities. That is why the combination of a Fuzzy Inference System and a Genetic Algorithm was chosen for this project.

Combining these two methods allowed me to apply evolutionary learning in the development of a Fuzzy Inference system. The FIS allows for smooth operation and application of a knowledge base while the GA is used to tune the membership functions and construct an effective rule base. Fuzzy control is superior to discrete control for this application because the game itself operates continuously. Applying fuzzy control allows for smooth operation for the player. Another advantage of Fuzzy Systems is their adaptability and robustness. This paper is exploring a system that is trained in a constrained environment then tested against more complicated scenarios. The goal of this is to demonstrate how fuzzy systems are more adaptable than other soft computing methods.

![Diagram

Description automatically generated]()

Figure 1 Schematic of Game Variables

# Related Work:

The genesis of this project occurred while listening to a lecture by Dr. Nick Ernest. Dr. Ernest gave a lecture on his defense system Alpha that was designed as part of his dissertation [1]. The system designed by Dr. Ernest applied Genetic Fuzzy Trees for the control of Unmanned Combat Aerial Vehicles (UCAV). The project was broad in scope but for this project I focused on the weapon control portion of his system. The system, Alpha, utilized 3 separate FIS’s a confidence level FIS, an individual weapon FIS, and a whole squadron weapon FIS. Dr. Ernest was controlling a whole squadron of UCAV’s, so his system took that into account. The system is much more complex than the one I designed but at its lowest level they are trying to do the same thing.

In Pickering 2020 [2], Genetic Fuzzy Tetris player is designed. The player applied three inputs to assess and determine the optimal placement for each Tetris block. The system was designed to use the Tetris game itself as part of the fitness function. This was the inspiration for the design of my own fitness function. The GA developed, assesses the fitness of each chromosome by playing the game itself.

# Methodology:

## 3.1: Genetic Algorithm

The first step in the design of the controller was focused on constructing Genetic Algorithm that would be used to tune membership functions and construct a rule base for part of the controller. The GA follows the structure of the Continuous Genetic Algorithm created by Dr. Seyedali Mirjalili [3] but used functions I coded to operate. The controller is built as a Cascading Fuzzy System trained by a Genetic Algorithm. The GA structure can be seen in *Figure 2*, and the Cascading Fuzzy System can be seen in *Figure 3*.

![Diagram

Description automatically generated]()

Figure Controller Structure

![Diagram, waterfall chart

Description automatically generated]()

Figure Fuzzy System Structure

The GA allows user defined inputs for the probability of population size and maximum generations as well as probability of crossover, mutation, and elitism. Their values, respectively, were 50, 100, 0.9, 0.3, and 0.05. The GA also allows user defined inputs for the number, upper, and lower bound of each gene segment in the chromosome. In total, there were 94 alleles for each chromosome in the population. The values for these can be seen in Table 1. The steering FIS as well as the fire FIS were held constant to keep the number of alleles to a reasonable amount.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Aim FIS** | | | | | |
|  | **Input MF's** | | **Output MF's** | | **Rules** | |
| **No. of Alleles** | 13 | 13 | 9 | 9 | 25 | 25 |
| **Upper Bound** | 32 | 32 | 5 | 10 | 3 | 3 |
| **Lower Bound** | 0 | 0 | -5 | 0 | 1 | 1 |

Table GA User Inputs

The GA provides parameters to different portions of the Aim FIS. The steering and firing are controlled by FIS’s with fixed parameters, as these are simple one DOF controllers. The GA is tasked with programming all the membership functions of the Aim FIS as well as the Rulebase. All these values are generated in the form of integers, between the given upper and lower bounds for each segment.

The GA initializes alleles between the given upper and lower bound, before these alleles can be fed into their FIS some of them require an operation be carried out on before application. For the aim FIS, the input MF’s are multiplied by 25 while the output MF’s are multiplied by 10. These membership functions are dealing with pixel values, which are inherently small, the multiplication step is used to boost the values to reasonable levels. After this multiplication step, the values for each individual membership function are sorted to ensure the parameters for each membership function are structured from lowest to highest.

The GA initializes a population, measures the fitness, then proceeds to the selection process. The selection process used the normalized fitness as well as randomly generated probabilities to select 2 chromosome parents for mutation. The GA then moves the parents on to the crossover function.

The crossover function includes two separate algorithms. The GA chooses which algorithm to use by generating a random number between 0 and 1, if the number is > 0.5 the GA uses a single point crossover where the parent is crossed over at a single index. If the number is <= 0.5, the GA chooses an index in each gene segment and crosses the entire segment between indexes. An illustration of this can be seen in Table 2. The idea behind this method is using one very modest crossover function and one very aggressive crossover function to diversify the genes as much as possible. There are 7 gene segments, so this flip would occur 7 times throughout the gene. Then a random number is evaluated against the probability of crossover. If the random number is below the probability the crossed child genes are moved on to the mutation function.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Allele No.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **Parent Index Before Crossover** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **Parent Index After Crossover** | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 |
| 2 | 2 | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 1 |

Table Aggressive Crossover Example

The mutation function randomly mutates one allele in each gene segment. An array of random indices is generated, one index in each gene segment. Then a random integer, within the upper and lower bounds of each segment is generated and replaces the allele at the given index. Again, a random number is generated and evaluated against the probability of mutation.

The final GA function is elitism. The elitism function carries a certain percentage of the mutated population over to the next generation. For this project, the elitism rate was kept relatively low at 0.05. This was to encourage maximum diversity among the populations as they were evaluated from generation to generation. The final attribute of the GA is the fitness determination. That will be covered in the next section.

## 3.2 Genetic Algorithm Fitness Determination

The GA evaluates chromosomes applying them to a fitness function and measuring their fitness. For this controller, the fitness function was derived from the game space invaders itself. The game takes in the chromosome and applies the alleles to the relevant FIS’s. It then plays the game to determine the fitness. The fitness is determined by the outcome of each laser that is fired by the player. The values of each laser outcome is then aggregated and scaled by the time it took to play the game. The values for each laser outcome can be seen in Table 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Laser Outcome** | Miss | Far Miss | Miss | Near Miss | Hit |
| **Fitness Value** | -50 | 10 | 20 | 30 | 100 |

Table Fitness Values

As referenced before, there were two training methods developed. One to train against single enemies, generated repetitively. And another system to play the game with ever increasing enemies. The first system played the game five times for each fitness evaluation. One enemy was generated for each evaluation. To increase the breadth of the training set, one enemy is randomly generated in each of five zones across the X-axis of the game play area. The purpose of this was to ensure the enemies spanned all the way across the screen while still generating them in random locations. Since the randomness of the generation was constrained, we can say these enemies were generated in a pseudo-random fashion. The laser outcomes are aggregated each session then divided by the time it takes to complete the level. Destroying the enemies sooner yields a higher fitness value.

The second system aggregates laser outcomes the same way the first system does. The difference is the system plays against an ever-increasing number of enemies generated at pseudo-random locations. Each time the level is completed, meaning all the enemies are destroyed, the number of enemies increases by 2. Similarly, the outcome is divided by the time it takes to complete each level then multiplied by the level number. The evaluation ends when five enemies have made it passed the player. The diagram for both training methods is shown in Figures 4 and 5.

![Chart, scatter chart, bubble chart

Description automatically generated]()

Figure Single Enemy Training Diagram

![Diagram

Description automatically generated]()

Figure Multi Enemy Training Diagram for Level 1 and 2

## 3.3 Cascading Fuzzy Systems

The controller is made up of three FIS’s structured in a cascading fashion, meaning the output of one FIS is the input for the next. The parameters of the aim FIS are all trained by GA, the parameters of the steer and fire FIS are fixed as these are simple one input one output systems.

The input for the steer FIS is the difference between the player angle and angle that intersects the aiming point generated by the aim FIS. The membership functions (MF’s) for these can be seen in Figure 6. This is a simple one input one output FIS tasked with steering the player. Fuzzy logic was applied here, after a simple controller was constructed and tested. The motion generated by the simple controller produced intermittent player motion while the Fuzzy controller produces smooth rotations for the player. The rule base for this FIS is simple, there are five membership functions for the input and output. The linguistic terms for the input and output MF’s are negative large (NL), negative small (NS), zero (ZE), positive small (PS), and positive large (PL).

The input for the fire FIS is also the difference between the player angle and angle that intersects the aiming point generated by the aim FIS. The membership functions for these can be seen in Figure 7. This is a simple one input one output FIS tasked with telling the player when to fire. Again, there are five input membership functions: negative large (NL), negative small (NS), zero (ZE), positive small (PS), and positive large (PL). There are only two output membership functions for this FIS, do nothing (DN) and fire (FR). The rule base for the steer and fire FIS can be seen in Table 4.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rulebase for steer FIS** | | | |  | **Rulebase for fire FIS** | | | |
| **if** | NL | **then** | NL |  | **if** | NL | **then** | DN |
| **if** | NS | **then** | NS |  | **if** | NS | **then** | DN |
| **if** | ZE | **then** | ZE |  | **if** | ZE | **then** | FR |
| **if** | PS | **then** | PS |  | **if** | PS | **then** | DN |
| **if** | PL | **then** | PL |  | **if** | PL | **then** | DN |

Table Rulebase for steer and aim FIS

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 6 steer FIS Membership Functions

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 7 fire FIS Membership Functions

The membership function parameters for the aim FIS are all tuned by GA. The inputs for the aim FIS are the X position and Y position of the enemy. The outputs of the FIS are the X coordinates and Y coordinates of a point to aim at for the enemy. The GA provides an aiming distance, in the X and Y direction, and those distances are added to the enemy position to make up the new aiming coordinates. The program is continuous, so these coordinates are constantly updated. There are five input membership functions for each input. There are three membership functions each output. This is a two input two output system. The Rulebase for the aim FIS is also constructed by GA. The rules all follow the structure:

And

The Rulebase is constructed to use integers as assignments for the output membership functions. Each output has 3 possible membership functions, so an integer of 1, 2, or 3 was generated as a representation of the rule. With this, a Pittsburgh learning approach was used to construct the Rulebase.

The last portion of the FIS to cover is the defuzzification process. All three FIS’s used a Centroid defuzzification method following the method by Ross (2010) [4].

![A picture containing diagram

Description automatically generated]()

Figure 8 Defuzzification Method

This completes the explanation of the problem-solving methodology; I will detail the results of GA training and Fuzzy controller in the next section.

# Results:

## Training the GFS using GA

The primary goal of this research was to develop an explainable artificial intelligence player for the game Space Invaders. The secondary goal was to compare training methods to determine which produced a better system. The GA for both training methods converged at 70 generations. The curves can be seen in Figure 9. It is important to note, these values have been divided by the time it took to complete the game. The unit of time used is “Frames”, it is not uncommon for this value to be > 3000 frames while the laser fitness value is less than < 1000. This imbalance in magnitude accounts for the relatively small changes in fitness evaluation. The fitness functions and convergence can be seen below.

Chart

Description automatically generatedChart, box and whisker chart

Description automatically generated

*Figure 9 Convergence*

The GA was used to train tune the membership functions and construct a Rulebase for the aim FIS. The system tuned 13 parameters for each input and 9 parameters for each output. The input fuzzification requires 15 parameters to fully define five membership functions. The first and last parameter for each input were fixed at 0 and 800, the minimum and maximum possible value. This was done to ensure the membership functions spanned the entire game play area. The membership functions that were generated were as follows.

Chart

Description automatically generatedChart

Description automatically generated

Chart

Description automatically generatedChart, diagram

Description automatically generated

Chart, diagram

Description automatically generatedChart, line chart

Description automatically generated

Chart, diagram

Description automatically generatedChart, line chart

Description automatically generated

The GA was also responsible for training the Rulebase for the aim FIS. There was a total of 25 rules that were necessary for both the X aiming point and Y aiming point. The GA used a Pittsburgh learning approach to learn the 50 required rules for the system.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **aim FIS X Input** | | | | |
|  | X Rules | **NL** | **NS** | **ZE** | **PS** | **PL** |
| **aim FIS Y Input** | **NL** | 2 | 1 | 2 | 3 | 2 |
| **NS** | 2 | 2 | 3 | 3 | 2 |
| **ZE** | 1 | 2 | 3 | 2 | 3 |
| **PS** | 1 | 2 | 2 | 1 | 2 |
| **PL** | 2 | 2 | 3 | 3 | 1 |
|  |  |  |  |  |  |  |
|  |  | **aim FIS X Input** | | | | |
|  | Y Rules | **NL** | **NS** | **ZE** | **PS** | **PL** |
| **aim FIS Y Input** | **NL** | 3 | 2 | 2 | 2 | 2 |
| **NS** | 1 | 1 | 2 | 2 | 1 |
| **ZE** | 1 | 1 | 3 | 3 | 3 |
| **PS** | 1 | 3 | 1 | 2 | 2 |
| **PL** | 2 | 3 | 1 | 2 | 1 |

Table Rulebase from GA for Method 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **aim FIS X Input** | | | | |
|  | X Rules | **NL** | **NS** | **ZE** | **PS** | **PL** |
| **aim FIS Y Input** | **NL** | 1 | 2 | 2 | 3 | 1 |
| **NS** | 2 | 2 | 2 | 3 | 2 |
| **ZE** | 2 | 2 | 2 | 1 | 2 |
| **PS** | 1 | 1 | 1 | 3 | 1 |
| **PL** | 2 | 3 | 3 | 3 | 3 |
|  |  |  |  |  |  |  |
|  |  | **aim FIS X Input** | | | | |
|  | Y Rules | **NL** | **NS** | **ZE** | **PS** | **PL** |
| **aim FIS Y Input** | **NL** | 3 | 2 | 1 | 1 | 1 |
| **NS** | 3 | 1 | 3 | 3 | 2 |
| **ZE** | 2 | 1 | 1 | 1 | 2 |
| **PS** | 2 | 3 | 1 | 1 | 1 |
| **PL** | 2 | 1 | 3 | 1 | 3 |

Table Rulebase for GA for Method 2

## Results for Training Method Comparison

The final result to detail is the comparison of training methods. To compare the methods, the game was played in 10 game increments with each respective gene generated by GA. For these games, the enemies were generated completely at random. There were no constraints on their X positions. This was repeated 3 times for a total of 30 games played. The fitness value data was collected and analyzed. The results from the game analysis can be seen below.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fitness Comparison for Game Session 1** | | | | | | | | | | | |
| **Game #** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **Total** |
| **Method 1** | 0.421 | 0.787 | 1.043 | 1.048 | 1.171 | 1.306 | 1.386 | 1.627 | 1.998 | 2.497 | **13.2849** |
| **Method 2** | 0.205 | 0.223 | 0.812 | 0.966 | 1.105 | 1.239 | 1.402 | 1.441 | 1.482 | 2.01 | **10.8859** |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Fitness Comparison for Game Session 2** | | | | | | | | | | | |
| **Game #** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **Total** |
| **Method 1** | 0.55 | 0.729 | 0.803 | 0.826 | 0.918 | 1.118 | 1.222 | 1.324 | 1.438 | 1.584 | **10.5116** |
| **Method 2** | 0.547 | 0.57 | 0.617 | 0.779 | 0.865 | 1.15 | 1.255 | 1.357 | 1.445 | 2.127 | **10.7131** |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Fitness Comparison for Game Session 3** | | | | | | | | | | | |
| **Game #** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **Total** |
| **Method 1** | 0.648 | 0.826 | 1.016 | 1.175 | 1.381 | 1.521 | 1.588 | 1.813 | 1.844 | 1.967 | **13.7796** |
| **Method 2** | 0.671 | 0.972 | 0.982 | 1.005 | 1.116 | 1.12 | 1.485 | 1.778 | 2.829 | 3.335 | **15.2922** |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Average** | **Total Sum** | **Max** |  |  |  |  |  |  |  |  |
| **Method 1** | 1.253 | 37.58 | 2.497 |  |  |  |  |  |  |  |  |
| **Method 2** | 1.229 | 36.89 | **3.335** |  |  |  |  |  |  |  |  |

Table Results from Game Analysis

# Conclusions:

For this paper a Genetic Fuzzy Player was constructed and trained with GA. Different training methods were also explored so I could compare different philosophies. This system was tested by playing Space Invaders a total of 30 times. From the game analysis in Table 7 we can see the Single Enemy training method, Method 1, performed better on average, but the Multi-enemy training method, Method 2, produced a fitness value approximately 25% higher than the other system. In fact, this value was near the optimal value reached during the training of GA.

From this we can see that it is possible to train a system with a constrained scenario and still achieve favorable results when the constraints are lifted. We can also see from the 1st and 3rd training session that the Single enemy training method produced results on par with the Multi-enemy training. Further demonstrating that a Fuzzy System is capable of performing outside of it’s training data set.

There are some valuable insights I’ve gained from this research and intend to further the development of this system with them. One issue with the training was the enemies always started at the top of the screen and were targeted from left to right. This prioritized those areas of the game arena and neglected others. This is realized in the chaotic nature of the input membership functions for the aim FIS. I think the enemies rarely reached portions of the inputs space in training and thus the system was not comprehensibly trained. An improvement on this could involve generating enemies in all portions of the screen and targeting them in different sequence to expand the input covered reached during the training.

Another issue was the targeting of the enemies. The list of enemies is updated continuously so any intelligent alteration to the targeting sequence would immediately throw the system into chaos because the basis of those intelligent targeting alterations was now completely different. This occurred because the rest of the GFS was integrated directly into the game code, so they all functioned together iteratively. This problem could be rectified by including a separate targeting system that was not integrated into the controls of the game itself. That way a decision could be made outside of the game and those parameters fed into the game, only changing when there was an alteration in some aspect of the enemy force.

Overall, the system developed performed very well. The goal of creating a system and testing training methods both yielded favorable results and this architecture will be expanded on for future research.

# References

|  |  |
| --- | --- |
| [1] | D. N. Ernest, *Genetic Fuzzy Trees for Intelligent Control,* Cincinnati, OH, 2015. |
| [2] | L. Pickering, "Genetic Fuzzy based Tetris Player," in *North American Fuzzy Information Processing Society*, (2020). |
| [3] | S. Mirjalili, "The Continuous Version of the Genetic Algorithm," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/67694-the-continuous-version-of-the-genetic-algorithm. |
| [4] | T. Ross, Fuzzy Logic with Engineering Applications, 3rd Edition, John WIley and Sons Ltd., 2010. |