**Player rating through fuzzy inference systems**

Thomas Rhodes

P16203335

IMAT 3406

Abstract

With competitive Esports steadily rising as an industry through multi-billion dollar games such as League of Legends, the desire to break into the scene as upcoming talent has only increased over previous years. This system aims to assist “support” players in League of Legends in improving by calculating a rank based on their performance in a match. This is achieved through the use of MATLAB’s fuzzy inference system (FIS) libraries to create five FISs, with 12 inputs, five outputs, and two systems for non-fuzzy calculations. The results produced by these systems could provide guidance to the success of a player in a match, as well as acting as a more rewarding variation of the system already present in League of Legends. However, additional statistics and further fine-tuning of each system would be necessary for the system to produce more helpful results.

Background

League of Legends is a Multiplayer Online Battle Arena (MOBA) that has been one of the most popular online games for years (Gaudiosi, 2012). Players typically take on one of the five roles presented in the game and develop their skill in that role, advancing as they improve through the games rating system. The FISs that I’ve created aims to provide a rating of the player’s performance for each match, based on a variety of statistics pulled from the game itself. This would act as a replacement of the games current systems, which currently employ a rating system that compares player performance to other players. This approach can result in disgruntled players however, as the requirements for a high rating constantly shift and are unpredictable. The system I’ve designed aims to make the ratings more predictable but still difficult. Currently, the system is designed for players that primarily play the “support” role, where the player will need to either heal their team, absorb damage for their team, or act as an additional damage dealer. Which priority they have depends on the sub-role their character plays, which characters being more tuned and aimed at fulfilling one of these priorities.

Literature review

Fuzzy logic is a logical system which provides a natural way of dealing with problems in which the source of imprecision is the absence of a distinct criteria on what constitutes a membership in a class, rather than the influence of random variables (Zadeh, 1965). This ability to represent vagueness in the definition of a class means fuzzy logic had a natural fit in the development of game AI. As such, fuzzy logic was officially introduced in game development in 1996 in the Game Developer Magazine (O'Brien, 1996). Since this, fuzzy systems have developed in game AI and are now considered one of the most useful techniques for game AI design, and takes up entire chapters in multiple books of game AI development (Pirovano, 2012).

Pirovano continues to discuss how fuzzy logic is utilised in AI for games, discussing numerous benefits and pitfalls of fuzzy logic (Pirovano, 2012). He discusses how the basis of fuzzy logic is simple and relies on basic Boolean logic, a prerequisite likely possessed by an AI developer. This in turn makes fuzzy logic and ideal candidate for introducing advanced AI in to a game with little effort. This concept runs parallel to my own experience developing my fuzzy system. Despite knowing little about MATLAB as a coding platform, illustrated later by lack of understanding of basic coding practises, the basics of implementing FISs quickly become clear. This is likely influenced by my own coding knowledge, but it allowed much greater freedom in designing a fuzzy system as the accessibility of fuzzy logic lends itself to being developed. That is, it can support being expanded and altered to suit the circumstances as it relies on very simple basis’. As such, I was able to modify the intuitive rule system to allow nested for loops to iterate through a create a large array of rules. Pirovano also suggests that the fuzzy membership functions can be defined using response curves that are usually already implemented within video games for simpler behaviours, again demonstrating the accessible nature of fuzzy logic

One of the pitfalls of fuzzy logic that Pirovano discusses is the issue of having multiple AI driven agents that need to iterate through a complex list of rules. This is that doing this iteration constantly can an adverse effect on game performance, which would be circumvented either through reduction of the agents or the rules, either restricting the max cap of intelligent enemies or undermining the completeness of the rule system. This is a factor that is prevalent in my system, which would ideally be integrated into a video game to provide players with a constant guide for improvement. If the system needed to iterate through thousands of rules, each needing to grab statistics from a server, while the client is also processing the end of the match, it could cause a massive drop in performance at the end of each game. This could prove frustrating for players having to wait, as well as actively detrimental on low performance machines as it could cause crashes. As such, in a final rendition of this fuzzy system, some form of hierarchical behaviours to resolve groups of rules at once and other applicable fixes, as suggested by Pirovano, would need implementation if this fuzzy system to function effectively.

Another application similar to mine is discussed by Zeng & Li in relation to football team ranking (Zeng & Li, 2014). In this they suggest the method of fuzzy clustering to compute the rank of 12 teams from a problem in the China Undergraduate Mathematical Contest in Modelling in 1993. In this they use clustering weighting to distinguish the rank of a team. This begins with simple calculations to decipher the total wins, loses, draws, and matches for each team, as well as their goals for, against, and their goal difference. They also use a fuzzy similarity matrix that compares the fuzzy similarity between the teams to determine the rankings. Using a variety of information, they determine that the team in the 12th place is team 4. From here, they find the next team that’s closest in similarity to team 4, team 5, and they take the place above. They continue to order the teams by their similarity to team 4, as team 4 sets the basis of what’s considered poor, and thus the further the similarity the better.

Their use of clustered weighting has clear benefits when used in ranking. It provides a clear order for how well you performed, which could be greatly beneficial in my system. It could be used to expand upon my current fuzzy system to allow it to track progress by using clustered weighting to rank games. This could help the player develop consistency by reviewing how they performed in their best and worst games, as well as demonstrate how they’ve developed over a longer period. It could be used to demonstrate similarities between a wide array of games they’ve won to point out their strong suits, and conversely to point out their weaknesses. However, while this system could be very helpful, it relies on comparison, which doesn’t help with determining a specific match’s rank, or with early games when there is still a small data sample. It also only achieves to match the predications made earlier in the report, thus questioning the validity of using fuzzy logic. When the results can be mathematically calculated and compared to rank them, how necessary does it become to use vague ideas to sperate them.

Overall, the literature suggests that fuzzy logic clearly has a purpose in video games and is actively beneficial to their design. It also demonstrates that fuzzy logic as means of providing a rating is effective, as it can help break down complex data.

Despite this, it also highlights flaws of fuzzy logic. In video games, the logic computation and the large rule sets can prove to be a burden on systems that are already struggling with graphics rendering and running the game engine. To be effective in video games, precautions and additional development is needed to ensure that the benefits out-weight the costs. As an approach for a rating system, fuzzy logic can fall short of purely mathematical systems, that use crisp data to provide more accurate data. In this case, fuzzy logic systems are merely a burden on problem, overcomplicating the process and obscuring the results.

In summation, fuzzy logic as a video rating system can work well, but has to be done carefully to prevent waste and confusion.

Design

System Overview

My fuzzy system utilises five FISs: Tank, Healer, Damage, Support Effectiveness, and Support Rank. It also contains two additional systems that are used in conjunction with the whole fuzzy system. The system works with an Excel document, “Support\_Rank.xls”, which stores both the input values used and the output values calculated. Each sheet within the Excel document represents a different system, containing the inputs and outputs for each. After each system, a for loop is used to write the output of a system both to its own sheet and the sheet of any system that requires that data.

The system begins with a mathematical system, which works out the percentage of the damage mitigated relative to the total damage taken. Following this, the Tank, Healer, and Damage FISs are defined and they evaluate the data. Then, a conditional system separates the Damage output from the others and evaluates which output from the Tank and Healer FIS is higher. This system then carries the appropriate value over to the Support Effectiveness FIS based on the sub-role of the character in the match. Support Effectiveness is then defined and using the carried over value, evaluates the data for the final FIS: Support Rank. This FIS is defined and evaluates to produce the final output.

See figure 1 in the appendices for a simple graphical representation of the system.

Fuzzy Inference Systems and other systems

FIS similarities

Most of the variables in my fuzzy system use the same design, and so to avoid repetition, they’ve been defined here to refer to. Any changes that are specific to that variable will be mentioned with the design.

Imported Design

The imported design has 5 membership functions: Very low, Low, Average, High and Very high. Very low and Very high both use a triangular membership function (trim MF), while the others use a gaussian membership function (gauss MF). All membership functions are evenly spaced apart, with both trim MFs being centred on the ends. The gauss MFs have a sigma that is equal to 10% of the variable’s range, and the Very low has the same for it’s range into the graph. The Very high is equal to 20% of the variables range.

I chose this arrangement with these ranges as it encourages Very High results, rewarding the player more and providing further recognition for exceptional performance. Also, I opted to make the sigma values and ranges a percentage of the variable range to ensure it remained even and consistent.

Forwarded Design

The forwarded design refers to any variable that has been calculated in this fuzzy system, such an output or an input that’s been carried forward from another FIS. The range of these variables is always 0 – 100. I chose this range as it is easy to understand for the user, as it can be interpreted as either a percentage or as a score, like traditional marking systems. This allows greater understanding of how one did. It also provided benefits for when I was making the system, as it kept things consistent between systems and meant it was easier to define the ranges fairly for each membership function.

This design also uses 5 membership functions: Very poor, Poor, Standard, Good, Very good. These all use gauss MFs, each with a range of 10% of the variable range, which is always 10 due to all variables under this design having a range of 100. These gauss MFs have the same spacing at the imported design.

Evaluation for loop

The evaluation for loop doesn’t refer to a design like the others, but is a feature present at the end of every FIS. This loop iterates through each row in the sheet for that FIS, and then evaluates the data for that row. This is done by referencing specific columns for each input, which would be written in prior or have been carried over by a previous FIS. Carrying over value to another sheet is also achieved in this same for loop, with it writing the output value from the evaluation to both the sheet it’s currently working from and the sheet that requires the data.

Mathematical system

This a very simple for loop that divides the mitigated damage statistic by the Total Damage Taken statistic to find the Mitigation Effectiveness. Mitigation Effectiveness represents the percentage of damage mitigated relative the total damage taken, and as such can be in excessive of 100%. This is because mitigated damage represents the amount of damage that was prevented, via effects like shields and armour, and so isn’t related to total damage, which is the total damage taken that effected the player’s health. It does this for each match present in Support\_Rank.xls, and then in the same for loop writes this to the Excel document.

Tank

The Tank FIS has two inputs, CC Score and Mitigation Effectiveness, and a single output, Tank Ability, and is used to evaluate the player’s performance as a tank.

CC Score

CC Score is an input used across three FISs, as it represents a key skill of a support’s performance regardless of sub-role. It represents how effective the player has been at applying “Crowd Control” (CC) to the enemy. CC is a category that represents impairing effects such as slows, stuns, freezes, and a variety of others. The input is measured on a scale of 0 to 75, as from my investigation this appears to be the very peak score most players achieve. This uses the imported design.

Mitigation Effectiveness

Mitigation Effectiveness is an input that is generated by the mathematical system detailed above. This has a range of 0 to 300, as from my research it would be exceptional difficult to achieve a Mitigation Effectiveness result in excess of 300%. This uses the imported design

Tank Ability

Tank Ability is the output from the Tank FIS and represents the players success as a tank in the match. This variable uses the forwarded design.

Rule Set

The rule set for the Tank FIS consists of 25 rules, written using my own knowledge. They all use the AND rule, and share a weighting of one. When constructing the rules, I aimed to ensure that both CC Score and Mitigation Effectiveness had the same impact, while also ensuring that the rules were somewhat forgiving. This was to try to reward success in either input. This is demonstrated in figure 2 in the appendices, an excerpt from the Tank FIS rule set, where in both rule 10 and rule 14 share the same Good output, apart from having a Low CC Score and an Average CC Score respectively.

Healer

The Healer FIS has two inputs, CC Score and Total Healing, and one output, Healer Ability, and is used to evaluate how well they performed as a healer.

CC Score

The input is near identical to the input with the same name in the Tank FIS, except that CC score has a reduced weight, which is translated through the rule set detailed below. This uses the imported design.

Total Healing

Total healing is the total amount of health the player has either gained or healed another player for. This variable has a range of 0 – 70000, as this provides a very high ceiling to prevent out of range values. This uses the imported design

Healer Ability

Healer Ability is the output from the Healer FIS and represents the players success as a healer in the match. This variable uses the forwarded design.

Rule Set

The rule set for the Healer FIS also consists of 25 rules, like the Tank FIS rule set. However, in this rule set, CC Score is less impactful, and can only have a positive impact. A player will always have at least the same Healer Ability output as their Total Healing input but can improve this by also having a higher CC Score. This is illustrated by the rules in figure 3 and figure 4 in the appendices from the Healer FIS: in figure 3, the output remains identical to the Total Healing input, while the output increased in figure 4 by the higher CC Score.

Damage

The Damage FIS has two inputs, CC Score and Total Damage, and one output, Damage Ability, and is used to see how successful the player was as a damage dealer.

CC Score

The input is the same as the Healer CC Score variable, and also uses the imported design.

Total damage

Total damage is the total amount of damage the player has inflicted on another player. This variable has a range of 0 – 100000, a value that from my research would be very difficult to obtain in a standard game. This uses the imported design.

Damage Ability

Damage Ability is the output from the Damage FIS and represents the players success as a damage dealer in the match. This variable uses the forwarded design.

Rule set

The rule set used by the Damage FIS is identical to the one used by the Healer FIS

Conditional system

This conditional system is used to decide what value is carried forward to the Support Effectiveness FIS as the Helpfulness output. It does this by first checking if the character is a “classic support”, either a healer or a tank, or a “DPS support”, a damage dealer. This is achieved by inputting either “sup” or “dps” in Support\_Rank.xls, which is then read in the system. If it’s a classic support, then it compares their support ability values, Healer Ability and Tank Ability, to see which is highest and carries it forward, and just carries the Tank Ability if they’re tied.

These work separately from the Damage Ability as if a healer support does a vast amount of tanking to the degree it gets a higher Tank Ability than its Healer Ability or vis versa for a tank support, that’s still acceptable. They’ve still performed well in their role as a classic support. However, if a healer or tank support has a higher Damage Ability than their respective ability, or a damage dealer has a high Tank or Healer Ability, this can be a sign of neglect and poor performance in their role. As such, it’s necessary to distinguish between them.

The final part is that if the support is a damage dealer, carry their Damage Ability forward.

Support Effectiveness

Support Effectiveness is the largest FIS in this fuzzy system, with 4 inputs: Helpfulness, Vision Score, Assists, and Deaths. It has a sole output, Support Rating, and is used to get an overall output of how the player performed with regards to other influential statistics.

Helpfulness

Helpfulness is an input generated by the conditional system above and represents the ability of the player in a specific sub-role. It uses the forwarded design.

Vision Score

Vision Score represents the player’s use of “wards” in the game. Wards are small, usually hidden items that allow players to see through the fog of war that block the map without their character being their directly. Their purchase and usage is typically the responsibility of the support, and as such is an important aspect of playing a support. Vision Score is a statistic that is calculated at the end of the game based on how effective the wards used by player have been. Vision Score has a range of 0-110, as the statistic tends to stay below this, preventing out of bounds results. This follows the imported design.

Assists

Assists is the total number of kills that the player has helped with, but not done the final blow to. They can acquire assists by doing one of these within a short period before the enemy is killed: damaging an enemy, healing an ally who damages or kills the enemy, applying CC to the enemy, granting vision to reveal the previously hidden enemy, and other case specific requirements. Assists has a range of 0-50, as it’s very unlikely for a team to achieve 50 kills in total, and thus even less likely for the support to acquire this many assists. This variable uses the imported design.

Deaths

Deaths is the total number of times that the player has died, providing the enemy team with resources whilst also leaving their whole team weaker, as they are now a member down until their can “respawn” after a timed delay. Deaths have a range of 0-30, as it’s unlikely the player would be able to die so many times before the game ends. Deaths uses the imported design.

Support Effectiveness

Support Effectiveness is an output that illustrates how well the player has performed as a support, accounting for the most important and influential statistics. This uses the forwarded design.

Rule set

The rule set for the Support Effectiveness FIS consists of 625 rules, which are generated through four nested for loops, all have a weight of one, and use the AND rule. Each for loop iterates one to five for an input value in the rule: “hlp” for Helpfulness, “vis” for Vision Score, “ast” for Assists, and “dth” for Deaths.. So, it’ll iterate through all rules in which the Helpfulness, Vision Score, and Assists are one, increasing the Death input value by one. Then the Assist value increases to two, and then it iterates through the Death for loop again. This repeats to complete a list of all 625 combinations of rules.

To then deduce the output value, it calculates a “suppValue”, which is the sum of the Helpfulness, Vision Score, and Assists input values subtracted by the Death input value. It then checks this value against If Else statements to decipher the Support Effectiveness output value.

The nested for loop structure and the suppValue calculation is shown in figure 5, with figure 6 showing some of the outputs from the fprintf in figure 5.

There are two additional values, “rCount” and “first”. rCount is a counter to track the rule number, used mainly for clarity when printing to the console. first is used as a Boolean value, to tell if it’s the first iteration. This is necessary as to concatenate a single rule onto ruleListD, ruleListD must first be initialised with a value. However, we don’t know what the first rule will be until we’re iterating through the nested for loops, and don’t want to be initialising the array with a bogus value. So, we use the first Boolean to initialise the array with the first rule before than continuing to concatenating normally.

Support Rank

This is the final FIS in the fuzzy system and is used to alter the end value relative to the game’s duration. It consists of two inputs, Support Rating and Game Duration, and one output: Support Rank.

Support Effectiveness

Support Effectiveness is detailed above in the Support Effectiveness FIS, but is now a input.

Game Duration

Game duration is an input with a range of 0-60 minutes, as this is representative of how long a game of League of Legends lasts. It’s made up of five membership functions: Very short, Short, Average, Long, and Very long. These are all represented by gauss MFs, whose sigma is 6, 10% of the variables whole range, and are spaced equally.

Support Rank

Support Rank represents the final output and result of the fuzzy system. This uses the forwarded design, but using a letter grading system for the membership function names, in this order from worst to best: D, C, B, A, S.

Rule set

The rule set for the Support Rank FIS is a rule set written using my knowledge, consisting of 25 rules, all with a weight of one and using the AND rule. It aims to be vaguely balanced, keeping the output average if the Game Duration input value and Support Effectiveness input value are the same, and increasing the output if the Support Effectiveness is greater than the Game Duration. However, it favours the player when the Game Duration is greater than the Support Effectiveness, illustrated in figure 7, where the decreasing and increasing of the output value is asymmetrical. This is to allow some leeway to reward the player more.

Evaluation of results

The fuzzy system produces results that clearly indicate the success of the player in a match. However, upon further testing and investigation, would need greater tuning to ensure that more minor changes in score can be accurately depicted. This could be altering the type of membership functions used, or by increasing the output values for some of the FISs, particularly the later ones.

The final result from the fuzzy system is show in figure 8, which highlights that, despite seemingly large differences in the data, the results don’t illustrate large differences in performance.

Alterations

The present fuzzy system went through major changes to reach the stage its at. Some of the key developments are listed below:

Using sheets in MATLAB

Originally, I intended to use multiple Excel documents, and limits the overall size of the fuzzy system to prevent the needing to use a large amount of documents. However, while reviewing the MATLAB documentation for “xlsread”, I realised I could utilise an inbuilt functionality of the function to write to specific sheets within a single Excel document. This allowed me to expand the idea further without worrying about having an excess of documents, as well as organise my data in an efficient manner.

Mathematical system

In its infancy, this fuzzy system took mitigated damage as its input for the Tank FIS over the current Mitigation Effectiveness. This was due to numerous factors: my lack of coding knowledge for MATLAB made the system seem more intimidating to create; not wishing to get too far ahead of myself before getting started on the heart of the project; lack of understanding of what the mitigated damage statistic meant in League of Legends. However, once the system had developed somewhat, I attempted to create the system to create a more representative value for the damage mitigated. It was only during this that I understood what mitigated damage represented in League of Legends, and that to refer to it without the context of total damage would be a huge mistake.

Conditional System

Originally, the conditional system was another FIS. Before I had improved my understanding of how the Fuzzy Logic Toolbox worked in MATLAB, I attempted to use a FIS that had five rules with the OR rule. The principle was to use the FIS to simply transfer the input value of whatever ability was highest, Support Ability or Tank Ability as this was prior to the inclusion of Damage Ability, as an output. However, this of course went through fuzzification and altered the output. So I replaced the whole FIS with a conditional system, now that I had attempted more conventional coding in the mathematical system, that transferred the highest value over. This ensured the value remained true through the process and reduced the overall computational overhead.

Introduction of damage dealers

For the majority of the system’s existence, it didn’t account for the whole sub-role of damage dealers, focusing only on tanks and healers. Only upon trying to gather data from the system to evaluate did I realise how many supports weren’t applicable due to this restriction. I had chosen to not include damage dealers originally because of limitations in my own understanding of how to utilise MATLAB. I had been unable to figure out an elegant method to differentiate the between a classic support and a damage dealer, as well as where to implement whatever system I used in the system. This was to prevent an issue detailed in the conditional system description in the design section of this report. However, after creating the conditional system to replace the previous FIS, I had a location to differentiate between the roles. I then investigated methods to input a difference between the roles. Through this, I learnt about how xlsread separates the data from the file you designate. It actually returns the data from the Excel document in three arrays: numerical data, text data, and raw data. It defaults to numerical data when not specified, which is an array of any cells containing only numbers, which is however I’ve usually interfaced with it. However, text data, returns an array of cells that contain any text and raw returns all data. With this, I can now specify in the Excel document whether the player is playing a classic support or a damage dealer through some simple for loops added onto the conditional statement. This let me expand the FIS to be entirely inclusive of the support role, adding much more validity to the whole fuzzy system.

Nested for loops to automatically generate rule set

When I first made the system, I designed it on paper and then implemented each fuzzy system first. This would allow me to set up the Excel document and order it in MATLAB first to have a guide. However, this meant I had no considered the implications of my design in regards to the rule set. It was only much later, that I realised that scale of the Support Effectiveness rule set, due to having four inputs each with five membership functions. This resulted in a total of 625 rules, which I initialled attempted to write by hand before giving up. I then planned to reduce the membership functions of each input, as the inputs themselves were too valuable to remove. However, I felt having only three membership functions per input would narrow the results too far and make them too vague to be helpful. So I instead opted to design a set of nested for loops to create the list for me.

While the for loops themselves and the overall concept was easy to implement, getting the system to work effectively took multiple attempts. I had issues with how to work out the correct output at first, as I needed a way for it to change dynamically based on the input values of each input. This originally took the form of a series of if statements to check each value, but proved to awkward and was altered into a single calculated value. While this did reduce my overall control of the rule system, it saved a lot of space, massively increased clarity, and helped with overhead as it iterated through the loops.

Another issue was trying to get the first check to work, and to correctly concatenate the rules onto the rule list. To begin with, I tried to concatenate two temporary rules, one that was from the first if statement, another for the current rule set being added. This was because of a lack of understanding in how concatenating worked, thinking that the array would be added to the end of the list. This method however just overwrote the original list with a list consisting of only the two previous rules. This was overcome when I added altered the concatenation to concatenate the temporary rule to the list, and then redefining the list as the new list with the now concatenated temporary rule.

I also attempted to use functions to save space and improve clarity. However, I struggled with the specific semantics of functions in MATLAB. I created the functions and they would have values passed through correctly when called to complete the function. Despite this, I struggled to get the values that were altered in the function to return to the main system. After struggling with this, I decided to forfeit functions, as they weren’t necessary for the functionality of the system, and would require a lot of time learning about how to structure and use functions which then wouldn’t translate into meaningful changes.

Conclusion

Overall I believe the fuzzy system achieves the aim of the project, provided a clear guide on how the player performed in their matches through the grading system. It accounts for a plethora of inputs and statistics from the game, stores all the data efficiently in a single Excel document, and utilises a wider functionality of MATLAB that I was unaware of before beginning. However, the system has stark limitations and would need much more in-depth tuning of individual values before being even near a professional system. Expansion of the system to account for more inputs, investigating and using a wider array of membership function types and defuzzification methods, and refined, delicate tweaking and testing would raise the system to a much more developed level.

Bibliography

Gaudiosi, J., 2012. *Riot Games' League Of Legends Officially Becomes Most Played PC Game In The World.* [Online]   
Available at: https://www.forbes.com/sites/johngaudiosi/2012/07/11/riot-games-league-of-legends-officially-becomes-most-played-pc-game-in-the-world/#7cc70885718b  
[Accessed 15 December 2018].

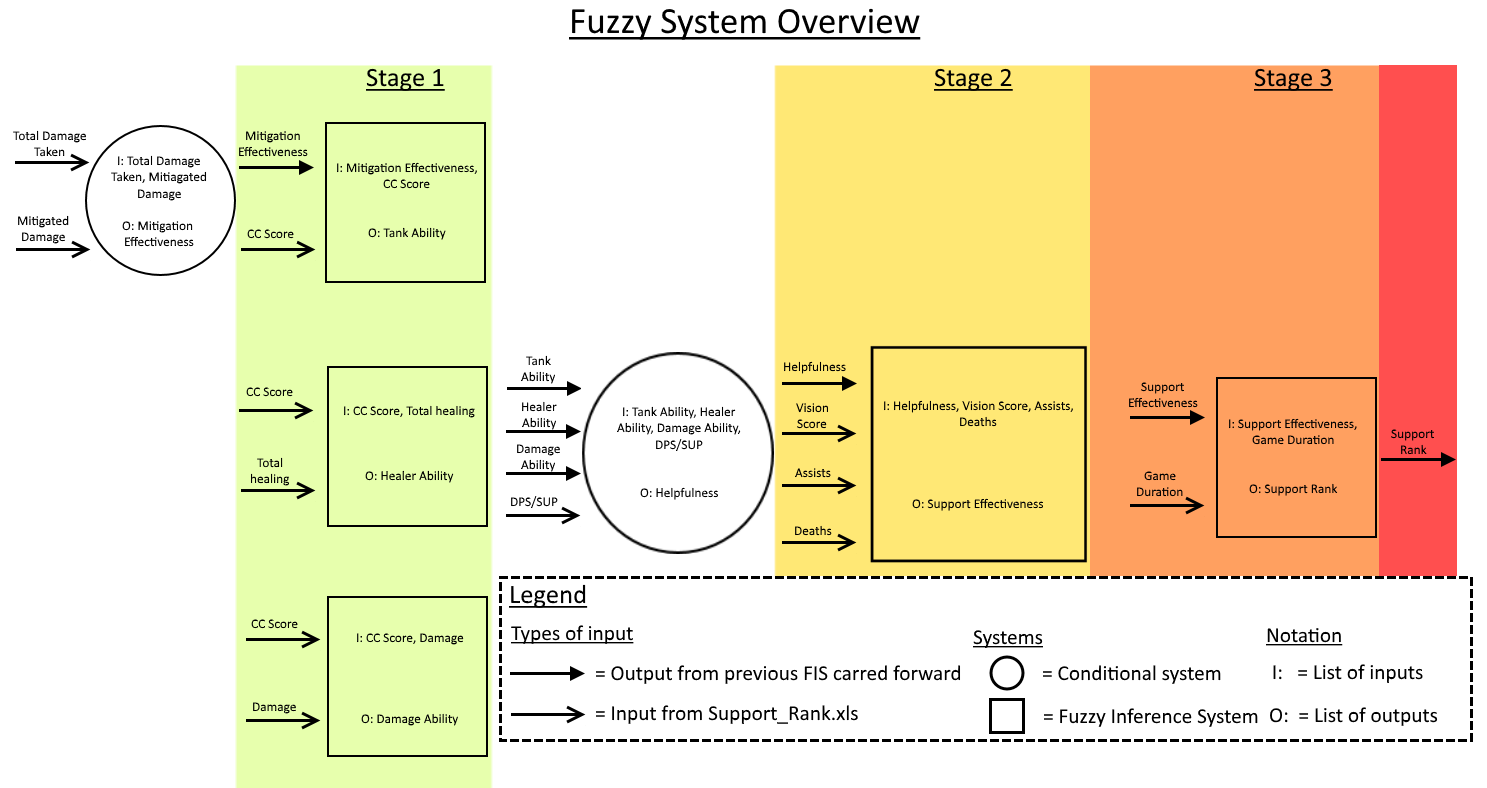
O'Brien, L., 1996. Fuzzy logic in games. In: L. O'Brien, ed. *Game Developer Magazine.* s.l.:Miller Freeman, Inc., pp. 52-55.

Pirovano, M., 2012. *The use of Fuzzy Logic for Artificial Intelligence in Games.* [Online]   
Available at: http://www.michelepirovano.com  
[Accessed 15 December 2018].

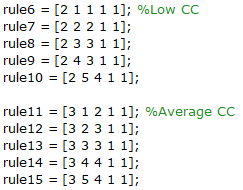
Zadeh, L. A., 1965. Fuzzy Sets. *Information and Control,* Volume 8, pp. 338-358.

Zeng, W. & Li, J., 2014. *Fuzzy Logic and Its Application in Football Team Ranking.* [Online]   
Available at: https://www.hindawi.com/journals/tswj/2014/291650/cta/  
[Accessed 15 December 2018].

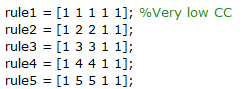
Appendices



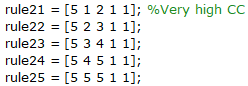
*Figure 1: Fuzzy System Overview*



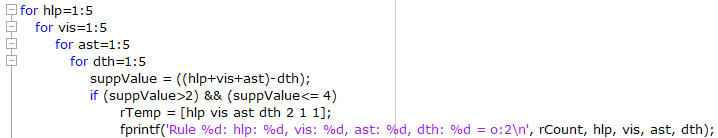
*Figure 2: Rules 6-15 of Tank FIS*

**

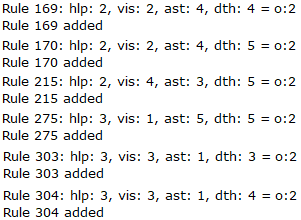
*Figure 3: Rules 1-5 of Healer FIS*



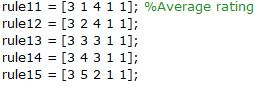
*Figure 4: Rules 21-25 of Healer FIS*

**

*Figure 5: Nested for loop structure and suppValue calculation*



*Figure 6: Rules from first if statement is nested for loop structure*

**

*Figure 7: Rules 11-15 of Support Rank FIS*