# Programming Assignment: Fuzzy Logic

## Topics and references

- Fuzzy sets. Membership function
- Fuzzy variables
- Fuzzy operators
- Fuzzy rules and inference system
- Fuzzification and Defuzzification

## Task

In this assignment you have to complete implementation of the Fuzzy Logic classes that can be used in development of games and simulations.

You can find the complete list of structures and functions to implement in form of pseudocode on the Moodle web page and in the assignment framework that is provided and accessible on the SVN server.

Complete instruction to the assignment and explanation of the method **will be given in the class**.

You implementation must passes all given tests in order to get the full mark for the assignment.

## Submission details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions. Since submission details will remain the same for all programming assessments, this portion will be skipped in future documents detailing labs and assignments.

### Source files

You have to submit the header `functions.h` and source file `functions.cpp`.

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <list>
#include <map>
#include <limits>
#include <cfloat> // FLT_EPSILON, ...
#include <memory> // shared_ptr


#include "data.h"

#define UNUSED(x) (void)x;

namespace AI
{
    // Comparison function for floats
    inline bool isEqual(float a, float b)
```

```
    {
        float epsilon = 128 * FLT_EPSILON;
        float abs_th = FLT_MIN;

        if (a == b) return true;

        float diff = std::abs(a - b);
        float norm = std::min((std::abs(a) + std::abs(b)),
std::numeric_limits<float>::max());

        return diff < std::max(abs_th, epsilon* norm);
    }


    //  Definition of the base fuzzy set class
    class FuzzySet
    {
    protected:
        // Members that define the shape
        float peakPoint;
        float leftOffset;
        float rightOffset;

        // representativeValue - the maximum of the set's membership function.
For instance, if
        // the set is triangular then this will be the peak point of the
triangular.
        // if the set has a plateau then this value will be the mid point of the
        // plateau. This value is set in creation to avoid run-time
        // calculation of mid-point values.
        float representativeValue;

        // DOM - holds the degree of membership of a given value in this set
        float DOM;

    public:
        FuzzySet(float peakPoint, float leftOffset, float rightOffset, float
representativeValue)
            : peakPoint{ peakPoint }, leftOffset{ leftOffset }, rightOffset{
rightOffset },
                representativeValue{ representativeValue }, DOM{ 0.0f }
        {
        }

        virtual ~FuzzySet()
        {
        }

        // calculateDOM - calculates and returns the degree of membership for a
particular value
        // NOTE, this does not set DOM to the value passed as the parameter.
        // This is because the centroid defuzzification method also uses this
method
        // to determine the degree of memberships of the values it uses as its
sample points.
        virtual float calculateDOM(float val) const
        {
            UNUSED(val);
```

```
            return 0.0f;
        }

        void clearDOM()
        {
            DOM = 0.0f;
        }

        float getDOM() const
        {
            return DOM;
        }

        void setDOM(float val)
        {
            DOM = val;
        }

        float getRepresentativeValue() const
        {
            return representativeValue;
        }

        // ORwithDOM - If this fuzzy set is part of a consequent floatiable,
        // and it is fired by a rule then this method sets the DOM (in this
        // context, the DOM represents a confidence level) to the maximum
        // of the parameter value or the set's existing member DOM value
        void ORwithDOM(float val)
        {
            if (val > DOM)
                DOM = val;
        }

        // Fuzzify a value by calculating its degree of membership
        FuzzySet* fuzzify(float val)
        {
            DOM = calculateDOM(val);
            return this;
        }
    };

    //  Definition of a fuzzy set that has a left shoulder shape.
    class FuzzySet_LeftShoulder : public FuzzySet
    {
    public:
        FuzzySet_LeftShoulder(float peakPoint, float leftOffset, float
rightOffset)
            : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint -
leftOffset / 2)
        {
        }

        // Your code ...

    };

    // Definition of a fuzzy set that has a right shoulder shape.
```

```
    class FuzzySet_RightShoulder : public FuzzySet
    {
    public:
        FuzzySet_RightShoulder(float peakPoint, float leftOffset, float
rightOffset)
            : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint +
rightOffset / 2)
        {
        }

        // Your code ...

    };

    // This defines a fuzzy set that is a singleton (a range
    // over which the DOM is always 1.0f)
    class FuzzySet_Singleton : public FuzzySet
    {
    public:
        FuzzySet_Singleton(float peakPoint, float leftOffset, float rightOffset)
            : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint)
        {
        }

        // Your code ...

    };

    // This is a simple class to define fuzzy sets that have a triangular
    // shape and can be defined by a mid point, a left displacement and a
    // right displacement.
    class FuzzySet_Triangle : public FuzzySet
    {
    public:
        FuzzySet_Triangle(float peakPoint, float leftOffset, float rightOffset)
            : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint)
        {
        }

        // Your code ...

    };


    // Fuzzy logic works with membership values in a way that mimics Boolean
logic.
    // Replacements for basic logic operators AND and OR are defined here.

    // Definition of the base operator class
    class FuzzyOperator
    {
    protected:
        std::list<std::shared_ptr<FuzzySet>> sets;

    public:
        FuzzyOperator(std::initializer_list<std::shared_ptr<FuzzySet>> sets =
{})
            : sets{ sets }
```

```
        {
        }

        virtual ~FuzzyOperator()
        {
        }

        // Returns the minimum DOM of the sets it is operating on
        virtual float getDOM()
        {
            return 0;
        }

        void clearDOM()
        {
            for (std::shared_ptr<FuzzySet> set : sets)
                set->clearDOM();
        }

        void ORwithDOM(float val)
        {
            for (std::shared_ptr<FuzzySet> set : sets)
                set->ORwithDOM(val);
        }
    };

    //  Definition of the AND operator class
    class FuzzyAND : public FuzzyOperator
    {
    public:
        FuzzyAND(std::initializer_list<std::shared_ptr<FuzzySet>> sets = {})
            : FuzzyOperator{ sets }
        {
        }

        // Your code ...

    };

    // Definition of the OR operator class
    class FuzzyOR : public FuzzyOperator
    {
    public:
        FuzzyOR(std::initializer_list<std::shared_ptr<FuzzySet>> sets = {})
            : FuzzyOperator{ sets }
        {
        }

        // Your code ...

    };

    // Definition of the fuzzy variable class
    class FuzzyVariable
    {
    protected:
        // A map of the fuzzy sets that comprise this variable
        std::map<std::string, std::shared_ptr<FuzzySet>> sets;
```

```
        // The minimum and maximum value of the range of this variable
        float minRange;
        float maxRange;

    public:
        FuzzyVariable()
            : sets{ }, minRange{ 0.0f }, maxRange{ 0.0f }
        {
        }

        virtual ~FuzzyVariable()
        {
        }

        std::shared_ptr<FuzzySet> getSet(const std::string& name)
        {
            return sets[name];
        }

        // This method is called with the upper and lower bound of a set each
time a
        // new set is added to adjust the upper and lower range values
accordingly
        void adjustRangeToFit(float minBound, float maxBound)
        {
            if (minBound < minRange)
                minRange = minBound;
            if (maxBound > maxRange)
                maxRange = maxBound;
        }

        // The following methods create instances of the sets named in the
method
        // name and add them to the member set map. Each time a set of any type
is
        // added the minRange and maxRange are adjusted accordingly.

        // Adds a left shoulder type set
        FuzzyVariable& addLeftShoulderSet(const std::string& name,
                                          float minBound, float peak, float
maxBound)
        {
            sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
                std::shared_ptr<FuzzySet>(new FuzzySet_LeftShoulder(peak,
                                          peak - minBound, maxBound - peak))));
            adjustRangeToFit(minBound, maxBound);
            return *this;
        }

        // Adds a left shoulder type set
        FuzzyVariable& addRightShoulderSet(const std::string& name,
                                           float minBound, float peak, float
maxBound)
        {
            sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
                std::shared_ptr<FuzzySet>(new FuzzySet_RightShoulder(peak,
                                           peak - minBound, maxBound - peak))));
```

```
            adjustRangeToFit(minBound, maxBound);
            return *this;
        }

        // Adds a triangular shaped fuzzy set to the variable
        FuzzyVariable& addTriangularSet(const std::string& name,
                                        float minBound, float peak, float
maxBound)
        {
            sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
                std::shared_ptr<FuzzySet>(new FuzzySet_Triangle(peak,
                                        peak - minBound, maxBound - peak))));
            adjustRangeToFit(minBound, maxBound);
            return *this;
        }

        // Adds a singleton to the variable
        FuzzyVariable& addSingletonSet(const std::string& name,
                                       float minBound, float peak, float
maxBound)
        {
            sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
                std::shared_ptr<FuzzySet>(new FuzzySet_Singleton(peak,
                                       peak - minBound, maxBound - peak))));
            adjustRangeToFit(minBound, maxBound);
            return *this;
        }

        // Fuzzify a value by calculating its degree of
        //  membership in each of this variable's subsets
        // takes a crisp value and calculates its degree of membership for each
set
        // in the variable.
        FuzzyVariable* fuzzify(float val)
        {
            //for each set in the flv calculate the degree of membership for the
given value
            for (std::pair<std::string, std::shared_ptr<FuzzySet>> set : sets)
                set.second->setDOM(set.second->calculateDOM(val));
            return this;
        }

        // Method for updating the DOM of a consequent when a rule fires
        void ORwithDOM(float val)
        {
            for (std::pair<std::string, std::shared_ptr<FuzzySet>> set : sets)
                set.second->ORwithDOM(val);
        }

        // Defuzzifies the value by averaging the maxima of the sets that have
fired.
        // Returns sum (maxima * degree of membership) / sum (degree of
memberships)
        float deFuzzifyMaxAv()
        {

            // Your code ...
```

```
            return 0.0f;
        }

        // Defuzzify the variable using the centroid method
        float deFuzzifyCentroid(int numSamples)
        {
            UNUSED(numSamples);

            // Your code ...

            return 0.0f;
        }
    };


    // Definition of the fuzzy rule class of form
    // IF antecedent THEN consequence
    class FuzzyRule
    {
    protected:
        // Antecedent (usually a composite of several fuzzy sets and operators)
        std::shared_ptr<FuzzyOperator> antecedent;

        // Consequence (usually a single fuzzy set, but can be several ANDed
together)
        std::shared_ptr<FuzzySet> consequence;

    public:
        FuzzyRule(std::shared_ptr<FuzzyOperator> antecedent,
                  std::shared_ptr<FuzzySet> consequence)
            : antecedent{ antecedent }, consequence{ consequence }
        {
        }

        // Updates the DOM (the confidence) of the consequent set with
        // the DOM of the antecedent set.
        std::shared_ptr<FuzzySet> calculate()
        {
            consequence->ORwithDOM(antecedent->getDOM());
            return consequence;
        }

        void setConfidenceOfConsequentToZero()
        {
            consequence->clearDOM();
        }
    };

    // Definition of the fuzzy module class
    class FuzzyModule
    {
    public:
        // Defuzzify methods supported by this module.
        enum DefuzzifyMethod { max_av = 0, centroid = 1 };

    protected:
```

```
        // When calculating the centroid of the fuzzy manifold this value is
used
        // to determine how many cross-sections should be sampled
        int numSamples;

        // A map of all the fuzzy variables this module uses
        std::map<std::string, FuzzyVariable> variables;

        // An array containing all fuzzy rules
        std::list<FuzzyRule> rules;

    public:
        FuzzyModule()
            : numSamples{ 15 }, variables{ }, rules{ }
        {
        }

        FuzzyVariable& getVariable(const std::string& name)
        {
            return variables[name];
        }

        // Zeros the DOMs of the consequents of each rule. Used by Defuzzify()
        void setConfidencesOfConsequentsToZero()
        {
            for (FuzzyRule rule : rules)
                rule.setConfidenceOfConsequentToZero();
        }

        // Creates and return a new 'empty' fuzzy variable.
        FuzzyVariable& createVariable(std::string varName)
        {
            variables.insert(std::pair<std::string,
                             FuzzyVariable>(varName, FuzzyVariable()));
            return variables[varName];
        }

        // Adds a rule to the module
        void addRule(std::shared_ptr<FuzzyOperator> antecedent,
                     std::shared_ptr<FuzzySet> consequence)
        {
            rules.push_back(FuzzyRule(antecedent, consequence));
        }

        // Calls the Fuzzify method of the variable with the same name
        void fuzzify(const std::string& varName, float val)
        {
            variables[varName].fuzzify(val);
        }

        // Given a fuzzy variable and a deffuzification method
        // this returns a crisp value
        float deFuzzify(const std::string& varName, DefuzzifyMethod method)
        {
            UNUSED(varName);
            UNUSED(method);

            // Your code ...
```

```
            return 0.0f;
        }
    };

} // end namespace

#endif
```

```
#include "functions.h"

namespace AI
{

} // end namespace
```

## Compiling, executing, and testing

Run `make` with the default rule to bring program executable `main.out` up to date:

```
$ make
```

Or, directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

To make a memory leak test, run `make` with target `leak`:

```
$ make leak
```

Make sure that the output does not show any memory related issue.

## File-level documentation

Every **edited by student** source and header file *must* begin with a *file-level* documentation block. This documentation serves the purpose of providing a reader the raison d'être of this source file at some later point of time (could be days or weeks or months or even years later). This module will use Doxygen to tag source and header files for generating html-based documentation. An introduction to Doxygen and a configuration file is provided on the module web page. Here is a sample for a C++ source file:

```
/*!*****************************************************************************
\file    functions.cpp
\author  Vadim Surov, <Your Name>
\par     DP email: vsurov\@digipen.edu, <Your Email>
\par     Course: CS380
\par     Section: A
\par     Programming Assignment 11
\date    04-30-2021

\brief
  This file has declarations and definitions that are required for submission
*******************************************************************************/
```

## Function-level documentation

Every function that you declare and define and submit for assessment must contain *function-level documentation*. This documentation should consist of a description of the function, the inputs, and return value. In team-based projects, this information is crucial for every team member to quickly grasp the details necessary to efficiently use, maintain, and debug the function. Certain details that programmers find useful include: what does the function take as input, what is the output, a sample output for some example input data, how the function implements its task, and importantly any special considerations that the author has taken into account in implementing the function. Although beginner programmers might feel that these details are unnecessary and are an overkill for assignments, they have been shown to save considerable time and effort in both academic and professional settings. Humans are prone to quickly forget details and good function-level documentation provides continuity for developers by acting as a repository for information related to the function. Otherwise, the developer will have to unnecessarily invest time in recalling and remembering undocumented gotcha details and assumptions each time the function is debugged or extended to incorporate additional features. Here is a sample for function `substitute_char`:

```
/*!*****************************************************************************
\brief
   Replaces each instance of a given character in a string with
   other given characters.

\param string
  The string to walk through and replace characters in.

\param old_char
  The original character that will be replaced in the string.

\param new_char
  The character used to replace the old characters

\return
  The number of characters changed in the string.
*******************************************************************************/
```

## Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit `functions.cpp` and `functions.h`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - `F` grade if your `functions.cpp` doesn't compile with the full suite of `g++` options.
    - `F` grade if your `functions.cpp` doesn't link to create an executable.
    - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. `A+` grade if output of function matches correct output of auto grader.
    - A deduction of one letter grade for each missing documentation block in `functions.cpp` and `functions.h`. Your submission `functions.*` (if it was edited by you)  must have **one** file-level documentation block and **one** function-level documentation blocks. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing or incomplete or copy-pasted (with irrelevant information from some previous assessment) block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an `A+` grade and one documentation block is missing, your grade will be later reduced from `A+` to `B+`. Another example: if the automatic grade gave your submission a `C` grade and the two documentation blocks are missing, your grade will be later reduced from `C` to `E`. Likewise, your submission `functions.h` must have **one** file-level documentation block and **one** function-level documentation block.