Algebra Intelligence Design Document

by
Bill Ezekiel and Lori Benson

Table of Contents

1. Introduction 3
2. Architecture 4
3. UML Diagram 5
    3.1 Question Class 6
    3.2 QuestionParser Class 6
    3.3 Polynomial Class 7
        3.3.1 Abs_Polynomial SubClass 7
        3.3.2 Power_Polynomial SubClass 8
        3.3.3 Future Polynomial SubClasses 8
    3.4 Term Class 9
    3.5 VarInfo Class 9
    3.6 User Class 9
    3.7 UserRating Class 9
4. Abstract Data Types 10
5. Database Design 10
6. Algorithm Design 11
7. Outside Sources 12

1. Introduction

Algebra Intelligence is a website designed to test a user’s knowledge in different categories of Algebra through a series of multiple choice quiz questions from each of the categories. The answers the users provide help to identify their strengths, weaknesses, and rank them as far as their skill level. After a question is answered a certain amount of times, a question’s skill level is adjusted based on the performance of the users.

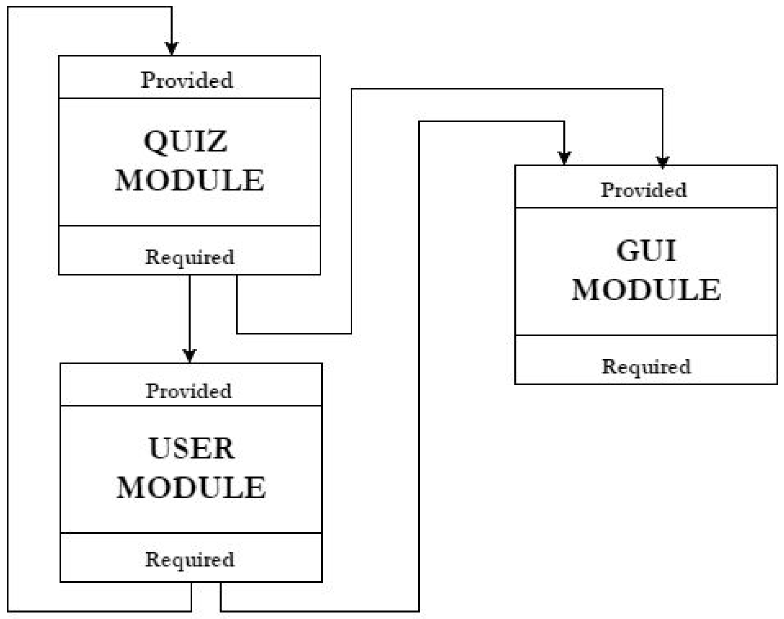
The main goal of Algebra Intelligence is to address the strengths and weaknesses of the users and to redirect them to other websites for additional help where it is needed. The website accomplishes this goal by automatically generating pre-defined questions of various categories and skill levels. The skill level of these questions corresponds to the user’s skill level in the question’s category. Upon completion of a quiz, users will receive an assessment ranking their overall performance. In categories defined as a user’s weakness, links to outside resources will be provided such that users can practice these skills.

The graphical user interface provides the user an easy to use website with simple point-and-click buttons. The buttons allow the user to either log in, or if they are a returning user, to start another quiz. The user log in button redirects them to another web page that requires a username and password to access the Algebra Intelligence quiz. The start quiz button allows the user to start a quiz, but only if the user is logged into the system.

The Algebra Intelligence quiz is composed of numerous questions from different categories based on different skill levels that are automatically generated from a question database. Based on a question’s template, a “solve” function is used to provide the correct answer to the question.

2. Architecture

The high level architecture of the system is shown below.

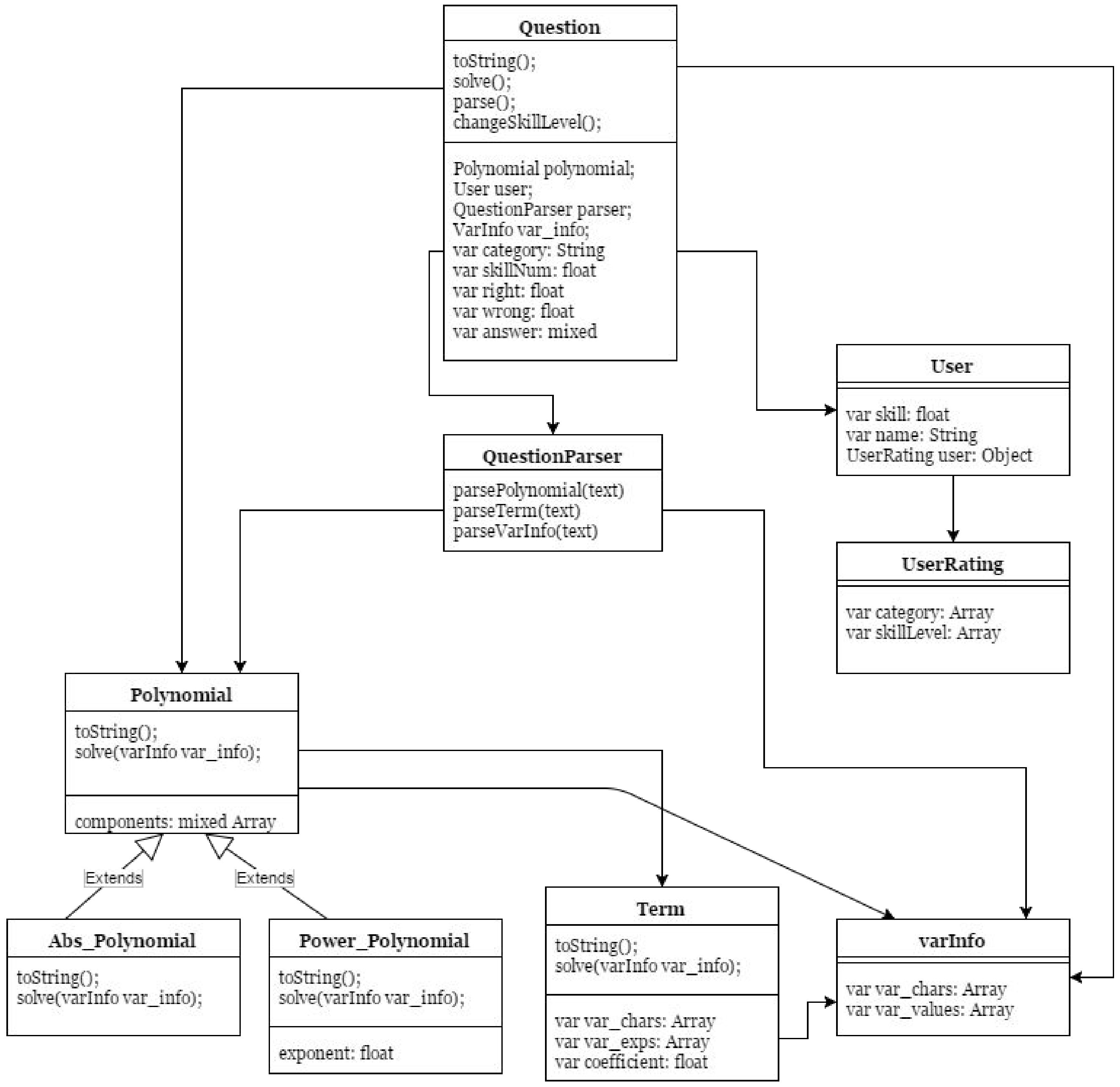


HIGH LEVEL ARCHITECTURE OF ALGEBRA INTELLIGENCE

The modules will be coded and implemented using the following languages:

Javascript - in charge of back-end processes.
HTML - web page design ,
MySQL - controls databases.
CSS - an appealing graphical user interface.

3. UML Diagram



3.1 Question Class

Purpose

The Question class is used to store information about a question. It stores the polynomial, skill level, number of correct and incorrect answers, category, the answer, and information about the variables in the polynomial. It uses information from the user to generate a question of appropriate skill level.

function toString()

Definition:	Converts the question to a string.
Returns:	A String representation of the question.

function solve()

Definition:	Solves the question using a given question template.
Returns:	The answer to the question. Instead of just pulling the answer from its 'answer' field it solves the polynomial by going through each term and sub polynomial and solving them.

function parse()

Definition:	Pull q_components and q_var_info from the question database and convert them into a Polynomial and VarInfo for the question.

function changeSkillLevel()

Definition:	Once a question has been answered a certain amount of times, change its skill level based on the number of correct and incorrect answers.

3.2 Question Parser Class

Purpose

The QuestionParser is responsible for taking strings and turning them into objects, specifically Polynomials, Polynomial's subclasses, Terms, and VarInfo. The parsed objects are then added to the Question class.

function parsePolynomial(Text)

Definition:	Parses JSON String to a Polynomial Object.
Parameters:	A String representation of a Polynomial.
Returns:	Returns a Polynomial Object
Errors:	If there are invalid text commands

function parseTerm(Text)

Definition:	Parses JSON String to a Term Object.
Parameters:	A String representation of a Term.
Returns:	Returns a Term Object
Errors:	If there are invalid text commands.

function parseVarInfo(Text)

Definition:	Parses JSON String to a VarInfo Object.
Parameters:	A String representation of the VarInfo.
Returns:	Returns a VarInfo Object
Errors:	If there are invalid text commands.

3.3 Polynomial Class

Purpose

The Polynomial class holds Terms and String representations of operators. These are collectively known as components, which the Polynomial uses to solve and display itself.

function toString()

Definition:	Converts a Polynomial to a string
Returns:	A String representation of the Polynomial.

function solve(var_info)

Definition:	Using variable values from a var_info object, evaluate the polynomial.
Parameters:	a VarInfo Object
Returns:	the evaluated polynomial. May be a boolean value or number depending on the type of question asked.

3.3.1 Abs_Polynomial SubClass

Purpose

A subclass of polynomial used to evaluate polynomials with absolute value. The form of these polynomials is: |*polynomial*|.

function toString()

Definition:	Converts the Abs_Polynomial to a string.
Returns:	A String representation of the Abs_Polynomial.

function solve(Object)

Definition:	Using variable values from a var_info object, evaluate the polynomial and then take its absolute value.
Parameters:	a VarInfo Object
Returns:	the evaluated polynomial. May be a boolean value or number depending on the type of question asked.

3.3.2 Power_Polynomial SubClass

Purpose

A subclass of polynomial used to evaluate polynomials taken to a power. The form of these polynomials is: (*polynomial*)^N, where N is the value of the *exponent* class variable.

function toString()

Definition:	Converts the Power_Polynomial to a string.
Returns:	A String representation of the Power_Polynomial.

function solve(Object)

Definition:	Using variable values from a var_info object, evaluate the polynomial and then take it to the Nth power.
Parameters:	a VarInfo Object
Returns:	the evaluated polynomial. May be a boolean value or number depending on the type of question asked.

3.3.3 Future Polynomial Subclasses

The reason for listing the Abs_Polynomial and Power_Polynomial subclasses was to prove that operations that can involve an entire polynomial (absolute value & polynomial to a power). This opens up the possibilities to more polynomial subclasses (e.g. SquareRoot_Polynomial).

3.4 Term Class

Purpose

The Term class represents a monomial within a polynomial. A term contains a coefficient, a list of variable letters (or characters), and a list of each variable's exponent. Like polynomials, terms can be converted to a String and solved.

function toString()

Definition:	Converts a Term to a string
Returns:	A String representation of the Term Object

function solve(var_info)

Definition:	Using variable values from a var_info object, evaluate the Term
Parameters:	A VarInfo object
Returns:	the evaluated value of this term.

3.5 VarInfo Class

Purpose

The VarInfo class consists of two arrays: variable characters (var_chars) and variable values (var_values). The point of this class is to save space, since these problems can become infinitely large. When there are an abundance of terms and only one variable 'y', it is more efficient to store the ['y'] and its value in VarInfo once as opposed to storing a thousand instances of it throughout each term.

3.7 User Class

Purpose

The purpose of the User class is to store information about the user. The data fields defined within the class are skill and name. The first is assigned a float data type and the second a String data type. A UserRating object is the last data field defined in the User class to obtain the user's rating in the defined category and the skill level.

3.8 UserRating Class

Purpose

The purpose of this class is to rate the user for each of the Algebra categories and also their skill level. The UserRating class contains one array of categories with the user’s matching skill level in that category in the same location of another array.

4. Abstract Data Types

Our software will utilize the MySQL database. The two databases will contain the following data types:

VARCHAR(X): String lengths up to 65,535 characters
MEDIUMTEXT(X): String lengths up to 16,777,215 characters
INT(X): Integer number value up to X digits.
FLOAT(X): Decimal based number value to X digits.

5. Database Design

User Database Design

Username	Hash password	User Ratings
billbob	a123hrtyswl683	‘{"categories":[“Addition”], “skillLevel”:[2.5]}’
VARCHAR()	VARCHAR()	MEDIUMTEXT

Question Database Design

Q_ID	Category	Q_Components	Q_Var_Info	Skill	Right	Wrong	Answer
1	Addition	‘{"components":[{"coefficient":2,"var_chars":[],"var_exps":[],"exp":1},"+",{"coefficient":2,"var_chars":[],"var_exps":[],"exp":1}]}’	‘{"var_chars":["x","y"],"var_values":[1,2]}’	1.5	7	3	“4”
INT	VARCHAR()	MEDIUMTEXT	MEDIUMTEXT	FLOAT	INT	INT	VARCHAR()

6. Algorithm Design

Solving polynomials, calling toString on polynomials, and parsing polynomials all require recursion to account for infinitely sized problems. The pseudocode for each process is given below.

Solving Polynomials

```
solve(varCharVal)
     resultString = “0”;
     for each component in the polynomial{
          if component is a term {
               resultString+= component.solve(varCharVal);
          }
          else if component is a Polynomial {
               //recursive call
               resultString +=component.solve(varCharVal)
          }
          else{ //used for Operators (e.g. “+”)
               resultString+= component
          }
     }
     return eval(resultString);
```

toString on Polynomials

```
toString()
     resultString = “”;
     for each component in the polynomial{
          if component is a term {
               resultString+= component.toString();
          }
          else if component is a Polynomial {
               //recursive call
               resultString += component.toString();
          }
          else{ //used for Operators (e.g. “+”)
               resultString+= component
          }
     }
     return resultString;
```

parsePolynomial

```
parsePolynomial(text)
      components = []
      var parsedText = JSON.parse(text);
      while(not at end of string){
            if(//found a term){
                  var termText = //part of text that is term.
                  var term = parse.Term(termText)
                  components.push(term)
            //end if}
            else if(//found a Polynomial){
                  var polyText = //part of text that is polynomial
                  var poly = parse.Polynomial(polyText)

                  if( //the polynomial was Abs_Polynomial){
                        poly = new Abs_Polynomial(poly.components)
                        components.push(poly)
                  // end if }

                  else if( // the polynomial was Power_Polynomial){
                        poly = new Power_Polynomial(poly.components)
                        components.push(poly)
                  //end else if}
                                   .
                                   .
                                   .
                     (Repeat for each Polynomial Subclass)
                  else{
                        components.push(poly)
                  //end else}
            // end else}
      //end while loop }
      var result = new Polynomial(components);
      return result;
```

7. Outside Sources

Algebra Intelligence utilizes the Bootstrap for the front-end development of the website. It handles the appearance of the site by establishing its own CSS classes. The site also uses Fractions.js, a javascript library, developed by Erik Garrison used to implement fractions in javascript. Questions for the Algebra Intelligence database are provided by the New York State Education Department and mrmcintoshsays.org.