Sudoku Game Design Language (SGDL)

Project Proposal

Team Members:

Sijue Tan	**st2669@columbia.edu**
Yigang Zhang	**yz2345@columbia.edu**
Yu Shao	**ys2528@columbia.edu**
Rongzheng Yan	**ry2213@columbia.edu**
William Chan	**wc2372@columbia.edu**

COMS 4115 Programming Languages and Translators

Professor Stephen A. Edwards

September 20, 2010   Columbia University

I. OBJECTIVE

To create a language that enables a programmer to quickly and easily design a Sudoku game.The language should be easy for programmer to manipulate the matrix and the element in it, to set rules for the game and to examine the rationality of the initialization. To make the application visual, we will output the simulation result on the screen. If it works well, we will popularize it to other word games design.

II. LANGUAGE DESCRIPTION

We were inspired by the AIC language from Andrew Willouer (Summer 2009) and the BOGuS language from Cary Maister (Spring 2009). SGDL's syntax is modeled after C/C++ and will be built on top of those languages. A SGDL format file will be the input file and the compiler will produce a C source file as output file that can then be compiled by any C/C++ compiler.

III. PROGRAM BREAKDOWN

A complete program must contain the following four sections that:

- defines the grid
- populates each cell in the grid
- defines the rules of the game, which the programmer does by calling methods of the grid
- develops a user interface, which must be console-/text-based

The program will start with a "main" declaration, similar to C/C++, followed by braces wrapping around the program statements. Each statement will end with a semicolon, and braces will be used to identify statement blocks. White spaces and carriage return characters will be used to separate tokens.

The compiler will test whether all the elements in the Sudoku grid conform to the rules as laid out by the programmer. The program will compile only if all the elements pass the test. If even one doesn't, the compiler will not compile the program and will instead produce a list of errors explaining where the problems are.

IV. PROGRAMMING FEATURES SPECIFICATION

Data Types:

- String
- Integer
- Boolean
- Grid

Objects:
SGDL contains only one object, the grid, which represents the Sudoku grid. It's defined with the syntax: *"Grid <gridname> (<row>, <column>);"*, and contains the following properties and methods:

Properties:

- Cell (<row#>, <column#>)
- Row (<row#>)
- Column (<column#>)
- Diagonal (<left | right>)
- Block (<row#>, <column#>): sub-square of the grid.
- Stack (<block#>)
- Brand (<block#>)

Methods:

- SetAsGiven (<false | true>): whether to show the value(s) in the specified cell(s). By default, all the cells in the grid are set to not visible.
- IsRepeatable (<false | true>): whether the values in the specified cells can be duplicates.
- DefineBlock (<rowsize>, <columnsize>): allows the programmer to define a sub-square of the grid. Its row and column dimensions must be factors of the grid's row and column dimensions respectively.
- SumOfRows (<integer>): the total that all the values in each row must sum to.
- SumOfColumns (<integer>): the total that all the values in each column must sum to.
- SumOfDiagonals (<integer>): the total that all the values in each diagonal must sum to.
- SumOfBlocks (<integer>): the total that all the values in each block must sum to.
- MaxValueOfRows (<integer>): what the maximum value of each row is permitted to be.
- MaxValueOfColumns (<integer>): what the maximum value of each column is permitted to be.
- MaxValueOfDiagonals (<integer>): what the maximum value of each diagonal is permitted to be.
- MaxValueOfBlocks (<integer>): what the maximum value of each block is permitted to be.
- MinValueOfRows (<integer>): what the minimum value of each row is permitted to be.
- MinValueOfColumns (<integer>): what the minimum value of each row is permitted to be.
- MinValueOfDiagonals (<integer>): what the minimum value of each diagonal is permitted to be.
- MinValueOfBlocks (<integer>): what the minimum value of each block is permitted to be.

Operators:

Arithmetic:

- ‘+’ (binary, addition)
- ‘-‘ (binary, subtraction)
- ‘*’ (binary, multiplication)
- ‘/’ (binary, division)
- ‘++’ (unary, postfix, first be used, then increments integer variable by one)
- ‘--‘ (unary, postfix, first be used, then decrements integer variable by one)

Comparison:

- ‘>’ (binary, greater than)
- ‘<’ (binary, less than)
- ‘=’ (binary, equal)
- ‘!=’ (binary, not equal)
- ‘>=’ (binary, greater than or equal to)
- ‘<=’ (binary, less than or equal to)

Logical:

- ‘not’ (unary, prefix)
- ‘and’ (binary)
- ‘or’ (binary)

Other:

- ‘.’ (binary, used only with grid object to access properties or methods)
- ‘is’ (binary, assignment)

Symbols

Brakets are used after while or if to state condition; used after methods to state the arguments; used after a cell to indicate its location
Commas are used to separate different arguments.
Braces are used for lists.
White spaces and carriage return characters will be used to separate tokens.
Double quotation marks are used to state a string.
Single quotation marks are used to state a character.

Commenting:

// (inline only)
/*.....*/ (All the characters between ‘/*’ and ‘*/’ will be treated as comments)

Keywords:
The following may only be used as keywords:

- main
- int
- bool
- string
- grid
- if-elseif-else
- for
- while
- not
- and
- or
- true
- false
- Print (<data type>)
- Scan (<data type>)
- Parse (<string>)
- Random (<seed>)

V. SAMPLE PROGRAM

```
main
{
   // Define the grid and populate each cell with the answers.
   Grid MyGrid ( 3, 3 ) is { 1, 2, 3,
                             3, 1, 2,
                             2, 3, 1 };

   // Define the rules:
   MyGrid.Cell ( 1, 2 ).MakeVisible ( true );
   MyGrid.Cell ( 2, 2 ).MakeVisible ( true );
   MyGrid.IsRepeatable ( false );
   MyGrid.SumOfRows ( 6 );
   MyGrid.SumOfColumns ( 6 );
   MyGrid.MaxValueOfRows ( 3 );
   MyGrid.MaxValueOfColumns ( 3 );
   MyGrid.MinValueOfRows ( 1 );
   MyGrid.MinValueOfColumns ( 1 );
```

```
    // Create the user interface.
    Print ( “Welcome to the Magic Squares Game! );
    Print ( MyGrid );

    string x;
    while ( x != “q” )
    {
        Print ( “Please identify the square by its row and column, then give its value (q to quit):  ” );
        x is scan ( string );
        if ( MyGrid.Cell ( Parse ( x, 1 ), Parse ( x, 2 ) ) = Parse ( x, 3 ) )
            Print ( “Right!” );
        else
            Print ( “Wrong!” );
    }
    ….

}
```

VI.EXTENTION

If time permits, we will try to extend the language further to enable programmers to develop word games such as word search puzzles and crossword puzzles.