BATCH B - TEAM 17

Elements Of Computing 2

Team Members

Kaushik Jonnada	CB.EN.U4AIE21122
Kalyana Sundaram	CB.EN.U4AIE21120
Sarvesh Shashi Kumar	CB.EN.U4AIE21163
Subikksha	CB.EN.U4AIE21167

TERM PROJECT

Design Predictive parser for Jack language that has the following grammar

expression:	term (op term)*
term:	integerConstant | stringConstant | keywordConstant | varName | varName '[' expression ']' | subroutineCall | '(' expression ')' | unaryOp term
subroutineCall:	subroutineName '(' expressionList ')' | (className | varName) '.' subroutineName '(' expressionList ')'
expressionList:	(expression (',' expression)*)?
op:	'+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='
unaryOp:	'-' | '~'
keywordConstant:	'true' | 'false' | 'null' | 'this'

Abstract:

- A parser for the JACK language built on python .
- The act of checking whether a grammar “accepts” an input text as valid is called parsing. As we noted
- earlier, parsing a given text means determining the exact correspondence between the text and the rules of a given grammar.
- Since the grammar rules are hierarchical, the output generated by the parser can be described in a tree-oriented data structure called a parse tree or a derivation tree.

What is Jack ?

High thoughts need a high language.

—Aristophanes (448-380 BC)

Designed to enable human programmers write high-level programs.Jack is a simple object-based language. It has the basic features and flavor of modern languages.

Just like Java and C#, with a much simpler syntax and no support for inheritance. In spite of this simplicity, Jack is a general-purpose language that can be used to create numerous applications.

In particular, it lends itself nicely to simple interactive games like Snake, Tetris, and Battle Ship

Standard Libraries in Jack

The Jack language comes with a standard library that may also be viewed as an interface to an underlying operating system. The library is a collection of Jack classes, and must be provided in every implementation of the Jack language. The standard library includes the following classes:

- **Math:** Provides basic mathematical operations;
- **String:** Implements the `String` type and basic string-related operations;
- **Array:** Defines the Array type and allows construction and disposal of arrays;
- **Output:** Handles text based output;
- **Screen:** Handles graphic screen output;
- **Keyboard:** Handles user input from the keyboard;
- **Memory:** Handles memory operations;
- **Sys:** Provides some execution-related services.

Math

- This class enables various mathematical operations.
- Function int abs(int x): Returns the absolute value of x.
- Function int multiply(int x, int y): Returns the product of x and y.
- Function int divide(int x, int y): Returns the integer part of the x/y.
- Function int min(int x, int y): Returns the minimum of x and y.
- Function int max(int x, int y): Returns the maximum of x and y.
- Function int sqrt(int x): Returns the integer part of the square root of x.

String

- This class implements the String data type and various string-related operations.
- Constructor String new(int maxLength): Constructs a new empty string (of length zero) that
- can contain at most maxLength characters.
- Method void dispose(): Disposes this string.
- Method int length(): Returns the length of this string.
- Method char charAt(int j): Returns the character at location j of this string.
- Method void setCharAt(int j, char c): Sets the j'th element of this string to c.
- Method String appendChar(char c): Appends c to this string and returns this string.
- Method void eraseLastChar(): Erases the last character from this string.
- Method int intValue(): Returns the integer value of this string (or at least of the prefix until
- a non numeric character is found).
- Method void setInt(int j): Sets this string to hold a representation of j.
- Function char backSpace(): Returns the backspace character.
- Function char doubleQuote(): Returns the double quote (") character.
- Function char newLine(): Returns the newline character.

Array

- This class enables the construction and disposal of arrays.
- Function Array new(int size): Constructs a new array of the given size.
- Method void dispose(): Disposes this array.

Output

- This class allows writing text on the screen.
- Function void moveCursor(int i, int j): Moves the cursor to the j’th column of the i’th row,
- and erases the character located there.
- Function void printChar(char c): Prints c at the cursor location and advances the cursor one
- column forward.
- Function void printString(String s): Prints s starting at the cursor location, and advances
- the cursor appropriately.
- Function void printInt(int i): Prints i starting at the cursor location, and advances the cursor
- appropriately.
- Function void println(): Advances the cursor to the beginning of the next line.
- Function void backSpace(): Moves the cursor one column back.

Screen

- This class allows drawing graphics on the screen. Column indices start at 0 and are left-to-right.
- Row indices start at 0 and are top-to-bottom. The screen size is hardware-dependant (over
- HACK: 256 rows * 512 columns).
- Function void clearScreen(): Erases the entire screen.
- Function void setColor(boolean b): Sets the screen color (white=false, black=true) to be
- used for all further drawXXX commands.
- Function void drawPixel(int x, int y): Draws the (x,y) pixel.
- Function void drawLine(int x1, int y1, int x2, int y2): Draws a line from pixel (x1,y1) to
- pixel (x2,y2).
- Function void drawRectangle(int x1, int y1, int x2, int y2): Draws a filled rectangle where
- the top left corner is (x1, y1) and the bottom right corner is (x2,y2).
- Function void drawCircle(int x, int y, int r): Draws a filled circle of radius r around (x,y)

Keyboard

- This class allows reading inputs from the keyboard.
- Function char keyPressed(): Returns the character of the currently pressed key on the keyboard; if no key is currently pressed, returns 0.
- Function char readChar(): Waits until a key is pressed on the keyboard and released, and then echoes the key to the screen and returns the character of the pressed key.
- Function String readLine(String message): Prints the message on the screen, reads the next line (until a newline character) from the keyboard, echoes the line to the screen, and returns its value. This method handles user backspaces.
- Function int readInt(String message): Prints the message on the screen, reads the next line (until a newline character) from the keyboard, echoes the line to the screen, and returns the integer until the first non numeric character in the line. This method handles user backspaces.

Memory

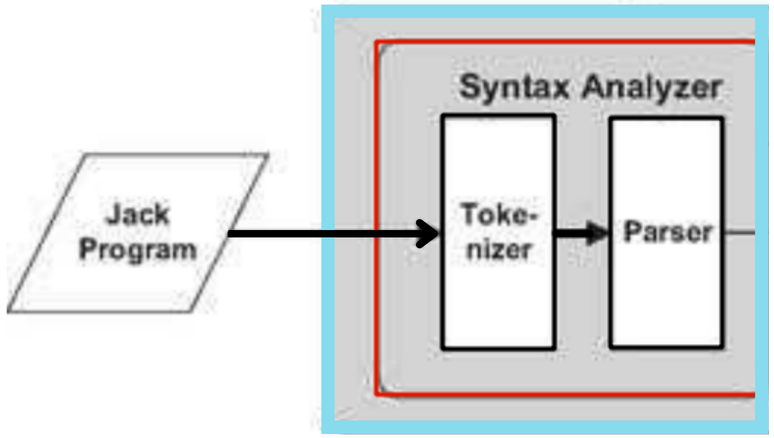
- This class allows direct access to the main memory.
- Function int peek(int address): Returns the value of the main memory at this address.
- Function void poke(int address, int value): Sets the value of the main memory at this address
- to the given value.
- Function Array alloc(int size): Allocates the specified space on the heap and returns a
- reference to it.
- Function void deAlloc(Array o): De-allocates the given object and frees its memory space.

Sys

- This class supports some execution-related services.
- Function void halt(): Halts the program execution.
- Function void error(int errorCode): Prints the error code on the screen and halts.
- Function void wait(int duration): Waits approximately duration milliseconds and returns.

Subroutine Calls

- The general syntax subroutineName is used in subroutine calls to invoke methods, functions, and constructors for their intended effects (argument-list).
- The subroutine's declared parameters, as well as the number and type of arguments passed in, must match. Even if the argument list is empty, the parentheses must still be present.
- Each argument could express any level of complexity. For instance, Jack's standard library's Math class has a square root function with the declaration function int sqrt (int n).
- Calls like Math.sqrt (17), Math.sqrt ((a * Math.sqrt (c- 17) + 3), and so forth can be used to call this function.



Parsing is the process of determining whether a grammar "accepts" a given input text as legitimate. As we mentioned before, parsing a text entails figuring out the precise correlation between the text and a specific grammar's rules. The output produced by the parser can be expressed in a tree-oriented data structure known as a parse tree or a derivation tree because the grammar rules are hierarchical.

Looking at the Code

SEM2-Assignments - PredictiveParser.ipynb

```
class PredictiveParser:
    def __init__(self, JackFileLoc):
        self.JackPath = JackFileLoc
        a = open(self.JackPath, "r")
        self.Code = a.readlines()
        self.kw = ['class','constructor','function','method','field','static','var','int','char','boolean','void','let','do','if','else','while','return','Array','String','Keyboard','screen']
        self.symbols = ['{','}','(',')','[',']','.',',',';','"']
        self.op = ['+','-','*','/','&','|','<','>','=']
        self.unop = ['~','-']
        self.integerconstant = [str(temp) for temp in range(0,3279)]
        self.keywordconstant = ['true','false','null','this']
        className = self.kw[0]
        self.stringconstants = ['Main']
        #make a new file uncommented code.txt
```

- We made a class called PredictiveParser
- We open the jack file in read mode.
- Each line is stored in self.Code.
- Self.kw,symbols,op,unop,integerconstant,keywordconstant all stores respective grammers. We check testing code with this as base.

SEM2-Assignments - PredictiveParser.ipynb

```
def rmvecmnts(self, commentedCode):
        a = open("UnCommentedCode.txt","w")
        p = 0
        for i in commentedCode:
            commentedCode[p] = i.split("//")[0]
            p += 1
        for i in commentedCode:
            commentedCode[commentedCode.index(i)] = i.split("/*")[0]
        for i in commentedCode:
            commentedCode[commentedCode.index(i)] = i.split("*/")[0]
        for i in commentedCode:
            commentedCode[commentedCode.index(i)] = i.split("*")[0]
        #concate the list to a string
        self.concode = "".join(commentedCode)
        a.write(self.concode)
        self.codeString = "\n"
        self.codeString = self.codeString.join(commentedCode)
        self.codeString = self.codeString.replace("{","\n{\n")
        self.codeString = self.codeString.replace("}","\n}\n")
        self.codeString = self.codeString.replace(";","\n;")
        self.codeString = self.codeString.replace(".","\n.\n")
        self.codeString = self.codeString.replace("(","\n(\n")
        self.codeString = self.codeString.replace(")","\n)\n")
        self.codeString = self.codeString.split()
        self.finalcode = "\n"
        self.finalcode = self.finalcode.join(self.codeString)
        #print(self.codeString)
```

- The first thing we do is removing comments.
- We make a file called "UnCommentedCode" and start writing to it.
- We go through each line in commentedCode(source Code) and splits if it encounters "//","/*","*","*/".
- Then if symbols are encountered we make new line before and after it so it becomes a new element in the list.
- Then after all this we join the whole uncommented stuff as self.CodeString which we will be using for comparing stuff.

SEM2-Assignments - PredictiveParser.ipynb

```
def xmltokenizer(self):
        self.tokenedcode = []

        temp1 =0
        for words in self.codeString:
                for keywords in self.kw:
                    if words == keywords:
                        self.tokenedcode.append("<keyword>" + words + "</keyword>")
                    else:
                        temp1 +=1
                for keywords in self.symbols:
                    if words == keywords:
                        self.tokenedcode.append("<symbol>" + words + "</symbol>")
                    else:
                        temp1 +=1
                for keywords in self.stringconstants:
                    if words == keywords:
                        self.tokenedcode.append("<stringConstant>" + words + "</stringConstant>")
                    else:
                        temp1 +=1
                for keywords in self.integerconstant:
                    if words == keywords:
                        self.tokenedcode.append("<intConstant>"+ words+"</intConstant>")
                    else:
                        temp1 +=1
                for keywords in self.op:
                    if words == keywords:
                        self.tokenedcode.append("<uniaryoperator>" + words + "</uniaryoperator>")
                    else:
                        temp1 +=1
                for keywords in keywords:
                    if words not in self.kw and words not in self.symbols and words not in self.stringconstants and words not in self.op and words not in self.integerconstant:
                        self.tokenedcode.append("<identifier>" + words + "</identifier>")
                    else:
                        temp1 +=1
        #print(self.tokenedcode)
```

- This is the betokening part where we tokenize each element from Self.CodeString .
- Basically we categorize each element as keyword,symbol,stringconstant, integerconstant and identifier.
- Logic is for each element in Self.CodeString if corresponding thing is found append it to respective category

SEM2-Assignments -

```
def symbolcheck(self):
        self.expressioncount =0
        for i in self.codeString:
            if i == '{':
                self.expressioncount += 1
            elif i == '}':
                self.expressioncount -= 1
            elif i == '[':
                self.expressioncount += 1
            elif i == ']':
                self.expressioncount -= 1
            elif i == '(':
                self.expressioncount += 1
            elif i == ')':
                self.expressioncount -= 1
        if self.expressioncount == 0:
            print("Symbols OK")
        elif self.expressioncount > 0:
            print("Symbols not closed")
```

- In this function we check if the count of symbol is equal to zero or greater than zero.
- Logic is , if count = 0 , all the symbols have its own closing correspondence .
- If not we inform the user that the symbols are not closed.

SEM2-Assignments - PredictiveParser.ipynb

```
def let(self):
    self.jlock = 0
    ssk = 0
    let = self.codeString
    for i in let:
        b = let[let.index(i)+1]
        if (i == "let"):
            # if self.symbols is present in b throw error
            if b in self.op:
                print("Let statement cannot have symbols after it")
                a = let.index("let")
                ssk = a
                for j in let[a + 1 : ]:
                    ssk += 1
                    if j == ";":
                        print(let[a:ssk])
                        self.jlock +=1
                        let[a] = "let "
                        break
                    self.jlock +=-1
            else:
                a = let.index("let")
                ssk = a
                for j in let[a + 1 : ]:
                    ssk += 1
                    if j == ";":
                        self.jlock +=1
                        let[a] = "let "
                        break
```

- The upcoming methods will be similar.
- The logic is if i = "let" , and there is no closing semicolon in the let statement we declare it as error.
- We also print that specific line so that the user can know which let statement has the error.
- This logic same for upcoming definitions with different conditions.

SEM2-Assignments - PredictiveParser.ipynb

```
def main(self):
        self.rmvecmnts(self.Code)
        self.dotprocessing()
        self.xmltokenizer()
        self.symbolcheck()
        self.var()
        self.let()
        self.ifcheck()
        self.whilecheck()
        self.methods()
        self.docheck()
        self.returncheck()
        print("Number of if statements: " + str(self.iflock))
        print("Number of else statements: " + str(self.elselock))
        print("Number of while statements: " + str(self.whilecount))
        print("Number of do statements: " + str(self.do))
        print("Number of return statements: " + str(self.returncount))
        print("Number of let statement: "+ str(self.jlock))
        print("Number of variable declarations: " + str(self.vardecloc))
```

- In this main function we call all the definitions we made in the class "PredictiveParser".
- We also print number of statements in the given code.

SEM2-Assignments -

```
a = PredictiveParser("SquareGame.jack")
a.main()
```

- In the snippet we call the class and assign it to a variable.
- We then call the main function of the class and which runs the definitions in the given order.

SEM2-Assignments - Main.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Average/Main.jack

// Inputs some numbers and computes their average
class Main {
    function void add() {
       var Array a;
       var int length;
       var int i;

       var int  sum;

       let length = Keyboard.readInt(" How many numbers? ");
       let a = Array.new(length); // constructs the array
       let  k = a + 1;
       let i = 0;
       while (i < length) {
          let a [ i ] = Keyboard.readInt(" Enter a number: ");
          let sum = sum + a [ i ];
          let i = i + 1;
       }

       do Output.printString(" The average is ");
       do Output.printInt(sum / length);
      return;
    }
 }
```

The Jack File used to test the parser.

Output of the parser which doesn't having comments

SEM2-Assignments - UnCommentedCode.txt

```
class Main {
    function void add() {
       var Array a;
       var int length;
       var int i;

       var int  sum;

       let length = Keyboard.readInt(" How many numbers? ");
       let a = Array.new(length);        let  k = a + 1;
       let i = 0;
       while (i < length) {
          let a [ i ] = Keyboard.readInt(" Enter a number: ");
          let sum = sum + a [ i ];
          let i = i + 1;
       }

       do Output.printString(" The average is ");
       do Output.printInt(sum / length);
      return;
    }
 }
```

Removed comments

```
['class', 'Main', '{', 'function', 'void', 'add', '(', ')', '{',
';', 'var', 'int', 'sum', ';', 'let', 'length', '=', 'Keyboard',
'a', '=', 'Array', '.', 'new', '(', 'length', ')', ';', 'let',
'<', 'length', ')', '{', 'let', 'a', '[', 'i', ']', '=', 'Keyboa
'sum', '=', 'sum', '+', 'a', '[', 'i', ']', ';', 'let', 'i', '='
'The', 'average', 'is', '"', ')', ';', 'do', 'Output', '.', 'pri
```

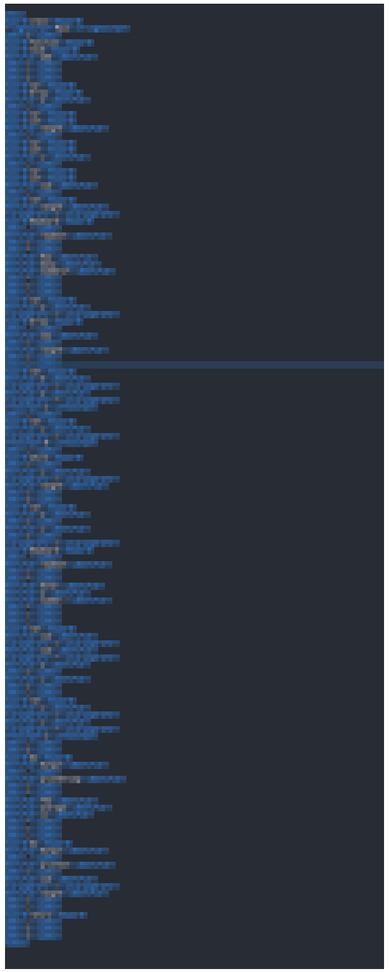
```
'var', 'Array', 'a', ';', 'var', 'int', 'length', ';', 'var', 'int', 'i',
'.', 'readInt', '(', '"', 'How', 'many', 'numbers?', '"', ')', ';', 'let',
k', '=', 'a', '+', '1', ';', 'let', 'i', '=', '0', ';', 'while', '(', 'i',
ard', '.', 'readInt', '(', '"', 'Enter', 'a', 'number:', '"', ')', ';', 'let',
, 'i', '+', '1', ';', '}', 'do', 'Output', '.', 'printString', '(', '"',
ntInt', '(', 'sum', '/', 'length', ')', ';', 'return', ';', '}', '}']
```

Self.CodeString

```
Symbols OK
Error: Variable name not declared
['let', 'k', '=', 'a', '+', '1']
[]
Number of if statements: 0
Number of else statements: 0
Number of while statements: 1
Number of do statements: 0
Number of return statements: 1
Number of let statement: 2
Number of variable declarations: 4
```

Output of the parser

As there is an undeclared variable it says the same!



xml output of tokenizer

Sneak Peak Code of Battle Ship

SEM2-Assignments - UnCommentedCode.txt

```
class Attack {

     static int rowInput, columnInput, score, hits;
     static Array ship1Row, ship1Column, ship2Row, ship2Column, ship3Row, ship3Column, ship4Row, ship4Column;

     function void setArrs(){
        do Attack.getshipRows();
        do Attack.getshipColumns();
        return;
     }

     function void askForAttack() {
          do Attack.setArrs();
          do Output.printString("Player 2 must guess the locations where ships are placed       ");
          let rowInput = Keyboard.readInt("Enter row number: ");
          let rowInput = 6 + (3 * (rowInput - 1));
          do Output.moveCursor(2, 0);
          let columnInput = Keyboard.readInt("Enter column number: ");
          let columnInput = 17 + (7 * (columnInput - 1));
          do Attack.compare();
          return;
     }
```

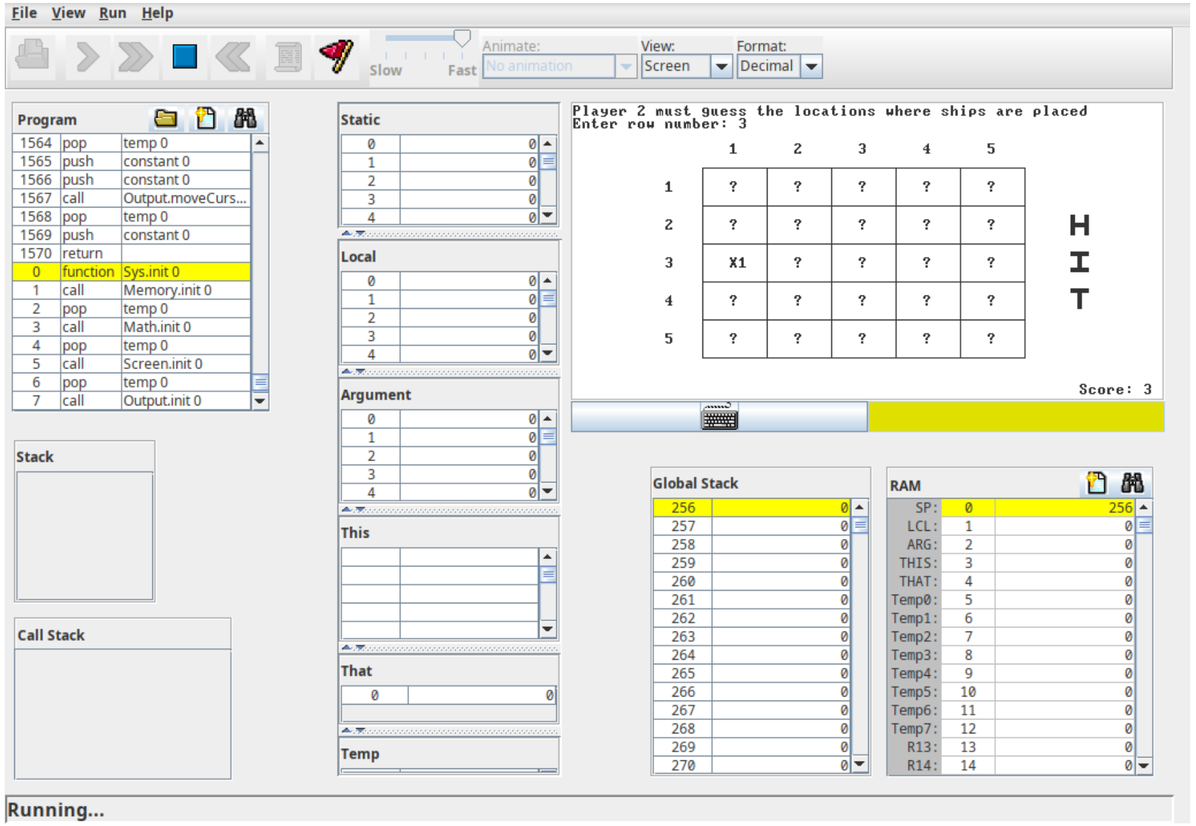
SEM2-Assignments - UnCommentedCode.txt

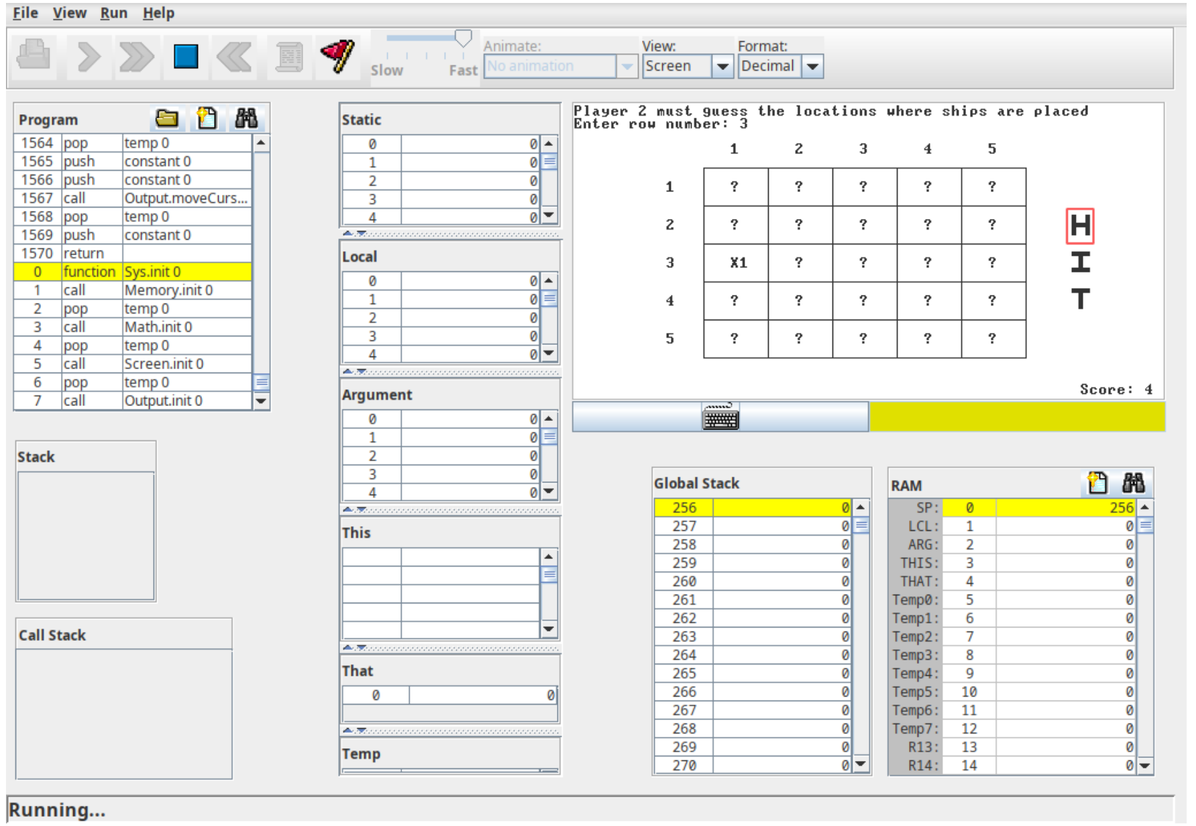
```
class Visual {

    static int memAddress, hStart, iStart, tStart, MStart, IStart, S1Start, S2Start;

    function void hitH() {
        let hStart = 3099;
        let MStart = 2587;
        let memAddress = 16384 + hStart;
        do Memory.poke(memAddress + 0, -4081);
        do Memory.poke(memAddress + 32, -4081);
        do Memory.poke(memAddress + 64, -4081);
        do Memory.poke(memAddress + 96, -4081);
        do Memory.poke(memAddress + 128, -4081);
        do Memory.poke(memAddress + 160, -4081);
        do Memory.poke(memAddress + 192, -4081);
        do Memory.poke(memAddress + 224, -1);
        do Memory.poke(memAddress + 256, -1);
        do Memory.poke(memAddress + 288, -1);
        do Memory.poke(memAddress + 320, -1);
        do Memory.poke(memAddress + 352, -4081);
        do Memory.poke(memAddress + 384, -4081);
        do Memory.poke(memAddress + 416, -4081);
        do Memory.poke(memAddress + 448, -4081);
        do Memory.poke(memAddress + 480, -4081);
        do Memory.poke(memAddress + 512, -4081);
        do Memory.poke(memAddress + 544, -4081);
        return;
    }
```

How the H of hit is made


File View Run Help
Slow Fast
Animate: No animation
View: Screen
Format: Decimal
Program
1564 pop temp 0
1565 push constant 0
1566 push constant 0
1567 call Output.moveCurs...
1568 pop temp 0
1569 push constant 0
1570 return
0 function Sys.init 0
1 call Memory.init 0
2 pop temp 0
3 call Math.init 0
4 pop temp 0
5 call Screen.init 0
6 pop temp 0
7 call Output.init 0
Static
Local
Argument
This
That
Temp
Stack
Call Stack
Player 2 must guess the locations where ships are placed
Enter row number: 3
1 2 3 4 5
1 ? ? ? ? ?
2 ? ? ? ? ?
3 X1 ? ? ? ?
4 ? ? ? ? ?
5 ? ? ? ? ?
HIT
Score: 3
Global Stack
RAM
SP: 0 256
LCL: 1 0
ARG: 2 0
THIS: 3 0
THAT: 4 0
Temp0: 5 0
Temp1: 6 0
Temp2: 7 0
Temp3: 8 0
Temp4: 9 0
Temp5: 10 0
Temp6: 11 0
Temp7: 12 0
R13: 13 0
R14: 14 0
Running...


File View Run Help
Slow Fast
Animate: No animation
View: Screen
Format: Decimal
Program
1564 pop temp 0
1565 push constant 0
1566 push constant 0
1567 call Output.moveCurs...
1568 pop temp 0
1569 push constant 0
1570 return
0 function Sys.init 0
1 call Memory.init 0
2 pop temp 0
3 call Math.init 0
4 pop temp 0
5 call Screen.init 0
6 pop temp 0
7 call Output.init 0
Static
Local
Argument
This
That
Temp
Stack
Call Stack
Player 2 must guess the locations where ships are placed
Enter row number: 3
1 2 3 4 5
1 ? ? ? ? ?
2 ? ? ? ? ?
3 X1 ? ? ? ?
4 ? ? ? ? ?
5 ? ? ? ? ?
HIT
Score: 4
Global Stack
RAM
SP: 0 256
LCL: 1 0
ARG: 2 0
THIS: 3 0
THAT: 4 0
Temp0: 5 0
Temp1: 6 0
Temp2: 7 0
Temp3: 8 0
Temp4: 9 0
Temp5: 10 0
Temp6: 11 0
Temp7: 12 0
R13: 13 0
R14: 14 0
Running...

CONTRIBUTIONS

M.Kalyana Sundaram - Parser

Kaushik Jonnada - Theory & Debugging

Sarvesh ShashiKumar - BattleShip

Subikksha - Theory & Debugging

THANK YOU