



华南师范大学
SOUTH CHINA NORMAL UNIVERSITY

课程：《编译原理课程项目》

所在学院：计算机学院

指导老师：王欣明

班级：软件技术五班

姓名：游森榕

2018 年 5 月

编译原理课程项目二

一、 编程语言

Java version 1.8.0_161

二、 开发工具

Eclipse

三、 实验内容

比特大战

1、游戏介绍

这个游戏来源于《自私的基因》⁴。一个 N 回合的比特大战由 A 和 B 两方进行，每个回合 A 和 B 可以选择 0（背叛）或 1（合作），共进行 N 个回合。如果第 $n(1 \leq n \leq N)$ 个回合 A 和 B 的选择分别为 A_n 和 B_n ，则他们在这个回合的得分 $S_A(n)$ 和 $S_B(n)$ 由下表决定：

	$B_n = 0$	$B_n = 1$
$A_n = 0$	$S_A(n) = 1, S_B(n) = 1$	$S_A(n) = 5, S_B(n) = 0$
$A_n = 1$	$S_A(n) = 0, S_B(n) = 5$	$S_A(n) = 3, S_B(n) = 3$

A 和 B 的总分为 N 个回合各自得分的和。

2、游戏的策略

这个游戏来源于《自私的基因》⁴。一个 N 回合的比特大战由 A 和 B 两方进行，每个回合 A 和 B 可以选择 0（背叛）或 1（合作），共进行 N 个回合。如果第 $n(1 \leq n \leq N)$ 个回合 A 和 B 的选择分别为 A_n 和 B_n ，则他们在这个回合的得分 $S_A(n)$ 和 $S_B(n)$ 由下表决定：

	$B_n = 0$	$B_n = 1$
$A_n = 0$	$S_A(n) = 1, S_B(n) = 1$	$S_A(n) = 5, S_B(n) = 0$
$A_n = 1$	$S_A(n) = 0, S_B(n) = 5$	$S_A(n) = 3, S_B(n) = 3$

A 和 B 的总分为 N 个回合各自得分的和。

T1 永远合作

```
Strategy T1: //T1表示策略的名字
return 1;
```

T2 随机

```
Strategy T2: //以0.75的概率返回1
i = RANDOM(3); //随机函数，RANDOM(k)以等概率返回0到k之间的一个整数
if (i == 3)
    return 0;
else
    return 1;
```

T3 针锋相对

```
Strategy T3:
//CUR表示当前的回合数
//A[1..N]和B[1..N]是两个预定义的数组分别保存自己和对手的每次选择
//在第CUR个回合，可以访问数组中的前CUR-1个元素
if (CUR == 1)
    return 1;
else
    return B[CUR-1];
```

T4 老实人探测器

```
Strategy T4:
if (CUR == 1)
    return 1;
else {
    i = RANDOM(9);
    if (i == 9)
        return 0
    else
        return B[CUR-1];
}
```

T5 永不原谅

Strategy T5:

```
i = 1;
k = 1;
while ((k < CUR) && (i == 1)) {
    if (B[k] == 0)
        i = 0;
    k = k + 1;
}
return i;
```

3、问题描述

- (B1) 设计一种程序设计语言可以用来描述比特大战的策略。这个语言应该至少可以描述上述 T1-T5 的策略，并用这个语言描述更多的策略。
- (B2) 实现对这个语言的语法分析。定义相应的语法树，并且可以输出语法分析的结果。当出现可能的输入错误时，可以指出出错的位置和可能的错误原因；
- (B3) 实现一个程序：用户可以输入若干的策略，每个策略保存为一个文本文件，模拟这些策略两两之间的 N 回合（例如， $n = 200$ ）的比特大战，并以所有对战得分的总和为这些策略排序；

四、 实验过程

1. 定义程序设计语言的上下文无关文法如下

(1) $\text{program} \rightarrow \text{function_declaration}$

程序由一个函数声明组成。

(2) $\text{function_declaration} \rightarrow \text{"Strategy"} \text{ ID } \text{statement_block}$

函数声明以字符串常量"Strategy"开头，接着是 ID(函数名)，最后是函数声明块 statement_block。

(3) $\text{statement_block} \rightarrow \text{"\{ " statement_list "}"$

函数声明模块由语句列表组成。

(4) $\text{statement_list} \rightarrow (\text{statement})^*$

语句列表由若干个语句组成。

(5) $\text{statement} \rightarrow \text{variable_declaration} \mid \text{assign_statement} \mid \text{if_statement} \mid \text{while_statement} \mid \text{return_statement} \mid \text{statement_block}$

一个语句可以是变量声明 (variable_declaration), 赋值语句 (assign_statement), if 语句 (if_statement), while 语句

(while_statement), return 语句(return_statement) 或者一个语句块(statement_block)

(6) assign_statement -> ID "=" expression ";"

赋值语句由变量名(ID), 赋值符(=), 表达式(expression) 和语句结束符(;) 组成。

(7) if_statement -> "if" "(" expression ")" [statement_block]

if 语句由"if", "(", 表达式(expression), ")", 语句块(statement_block) 组成。当条件表达式为假的时候, 语句块(statement_block) 被跳过。

(8) while_statement -> "while" "(" expression ")" statement_block

while 语句由"while", "(", 表达式(expression), ")", 语句块(statement_block) 组成。当条件表达式为假的时候, 语句块(statement_block) 被跳过。

(9) return_statement -> "return" expression;

return 语句由"return", 表达式(expression) 组成。

(10) expression -> logic_expression | <RANDOM> "(" <CONSTANT> ")" | ";"

表达式由逻辑表达式(logic_expression), 或者 random() 函数, 或者分号组成。

(11) logic_expression -> relational_expression("(" || " | "&&"

relational_expression)*

逻辑表达式(logic_expression) 由若干关系表达式(relational_expression) 进行与或关系运算组成。

(12) relational_expression -> additive_expression("<" | "<=" | "==" | ">=" | ">" | "!=") additive_expression)*

关系表达式(relational_expression) 由若干加/减法表达式(additive_expression) 进行大小运算组成。

(13) additive_expression -> multiplicative_expression("+" | "-"

multiplicative_expression)*

加法表达式(additive_expression) 由若干乘/除法表达式(multiplicative_expression) 进行加减法运算组成。

(14) multiplicative_expression -> factor("*" | "/") factor)*

乘法表达式(multiplicative_expression) 由若干算数因子(factor) 进行乘/除运算组成。

(15) factor -> "(" expression ")" | <ID> | <ID> "[" <CONSTANT> "]" |

<CONSTANT>

算术表达式因子(factor)包括由括号括起来的表达式, 变量名, 数组值, 常数。

(16)<LETTER> -> [a-zA-Z]

大写和小写的任意字母。

(17)<DIGIT> -> [0-9]

任意个位数

(18)<ID> -> (<LETTER> | _)(<LETTER> | _ | <DIGIT>)*

以下划线或字母开头的包含若干字母数字和下划线的字符串, 用于做函数名

2. 根据设计的程序设计语言, 使用 Javacc 以及 Javacc 中的 JTree 编写 .jjt 文件以及 .jj 文件。

在实验中主要有 YousBitWarJTreeGrammar.jjt, 其中定义了对设计的语言进行语法分析并构造语法分析树的规则, 编译 YousBitWarJTreeGrammar.jjt, 便会生成 YousBitWarJTreeGrammar.jj 文件以及 JJTYousBitWarJTreeState.java, Node.java, SimpleNode.java, YousBitWarJTreeTreeConstants.java 文件。

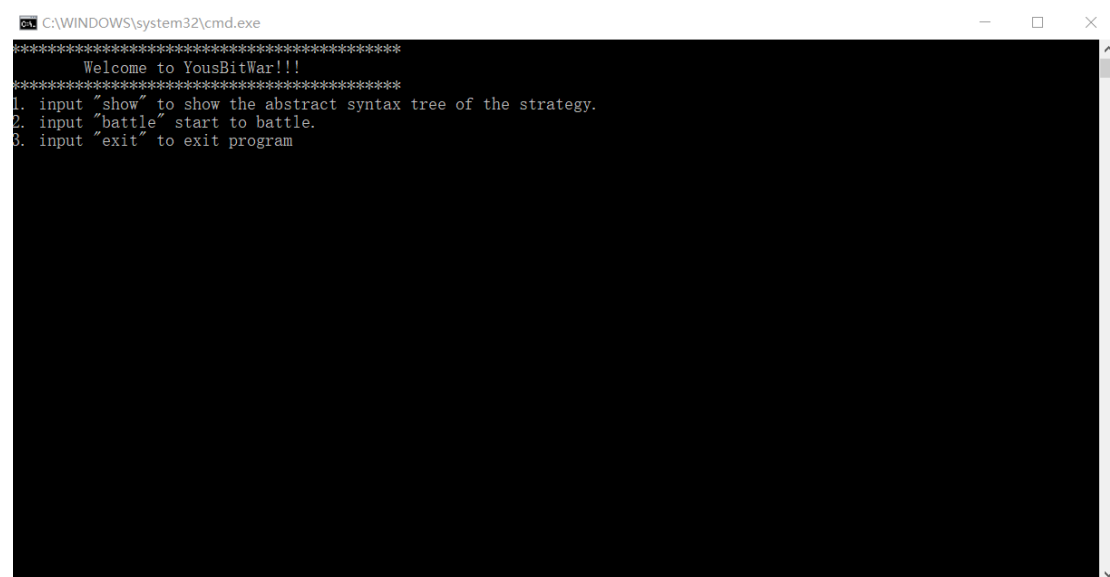
然后对 YousBitWarJTreeGrammar.jj 文件进行处理便会生成 SimpleCharStream.java, Token.java, TokenMgrError.java, YousBitWarJTree.java, YousBitWarJTreeConstants.java, YousBitWarJTreeTokenManager.java。

对于 YousBitWarGrammar.jj 文件, 在其中加入了语义动作, 用于对使用新设计的语言进行语义动作的执行, 从而达到运行设计的策略程序的目的。编译 YousBitWarGrammar.jj 生成 YousBitWarParser.java, YousBitWarParserConstants.java, YousBitWarParserTokenManager.java 等文件, 用于对策略程序进行语义分析。

3. 根据编写的 .jjt 和 .jj 文件以及这些文件生成的 java 文件, 编写一些文件调用生成的 java 文件中的类, 从而构造出整个完整的比特大战程序。
ExternData.java 文件中定义了 YousBitWarParser.java 中使用到的额外的数据结构;
YousBitWarJTree.java 定义了对策略描述程序进行语法分析, 以及构建语法分析树的类;
YousBitWarParser.java 定义了对策略描述程序进行语义动作执行的类;
YousBitWarPlayer.java 定义了一个比特大战中的作战的成员类;
YousBitWarBattle.java 定义了两个策略进行若干次大战的类;
YousBitWarDemo.java 定义了整个比特大战程序进行用户交互的类;
4. 使用设计的程序设计语言描述的比特大战策略存储于项目的 strategy 目录之中。

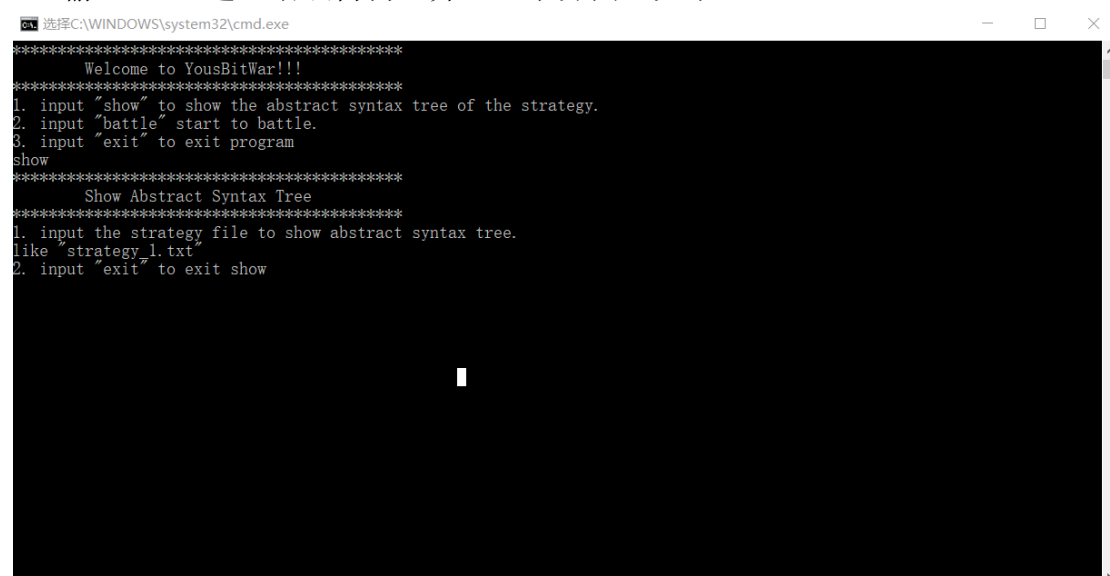
五、 实验结果

1. 通过双击 run. bat, 可以启动程序。程序界面如下:



```
C:\WINDOWS\system32\cmd.exe
*****
Welcome to YousBitWar!!!
*****
1. input "show" to show the abstract syntax tree of the strategy.
2. input "battle" start to battle.
3. input "exit" to exit program
```

2. 输入 show 进入语法树构造并且显示界面, 如下:



```
选择C:\WINDOWS\system32\cmd.exe
*****
Welcome to YousBitWar!!!
*****
1. input "show" to show the abstract syntax tree of the strategy.
2. input "battle" start to battle.
3. input "exit" to exit program
show
*****
Show Abstract Syntax Tree
*****
1. input the strategy file to show abstract syntax tree.
like "strategy 1.txt"
2. input "exit" to exit show
```

输入策略文件名, 可得分析树如下:

```
C:\WINDOWS\system32\cmd.exe
*****
      Show Abstract Syntax Tree
*****
1. input the strategy file to show abstract syntax tree.
like "strategy_1.txt"
2. input "exit" to exit show
strategy_1.txt
program
  function_statement
  statement_block
  statement_list
  statement
  return_statement
  expression
  logic_expression
  relation_expression
  additive_expression
  multiplicative_expression
  factor
Thank you.
*****
      Show Abstract Syntax Tree
*****
1. input the strategy file to show abstract syntax tree.
like "strategy_1.txt"
2. input "exit" to exit show
```

3. 输入 battle 进入比特大战模式，显示如下：

```
C:\WINDOWS\system32\cmd.exe
*****
      Welcome to YousBitWar!!!
*****
1. input "show" to show the abstract syntax tree of the strategy.
2. input "battle" start to battle.
3. input "exit" to exit program
show
*****
      Show Abstract Syntax Tree
*****
1. input the strategy file to show abstract syntax tree.
like "strategy_1.txt"
2. input "exit" to exit show
exit
*****
      Welcome to YousBitWar!!!
*****
1. input "show" to show the abstract syntax tree of the strategy.
2. input "battle" start to battle.
3. input "exit" to exit program
battle
*****
      Battle
*****
1. input the number of round to battle.
like: 100
2. input "exit" to exit show.
```

输入大战的回合数，即可进行大战，并且显示各个策略对战的结果，如下：


```
C:\WINDOWS\system32\cmd.exe
battle
*****
          Battle
*****
1. input the number of round to battle.
like: 100
2. input "exit" to exit show.
200
strategy_1.txt VS strategy_2.txt      384:744
strategy_1.txt VS strategy_3.txt      600:600
strategy_1.txt VS strategy_4.txt      537:642
strategy_1.txt VS strategy_5.txt      600:600
strategy_2.txt VS strategy_3.txt      525:525
strategy_2.txt VS strategy_4.txt      441:541
strategy_2.txt VS strategy_5.txt      74:739
strategy_3.txt VS strategy_4.txt      216:221
strategy_3.txt VS strategy_5.txt      600:600
strategy_4.txt VS strategy_5.txt      208:203
strategy_1.txt Score: 1737
strategy_3.txt Score: 1341
strategy_2.txt Score: 1040
strategy_5.txt Score: 882
strategy_4.txt Score: 865
*****
          Battle
*****
1. input the number of round to battle.
like: 100
2. input "exit" to exit show.
```

六、 展望

该程序是通过使用 Javacc 工具在 jj 文件中嵌入语义动作然后直接执行程序，后期希望能采用采用其他方法，把程序转换成 java 语言或者 C 语言，然后再转化成汇编语言，然后使程序运行起来。