Traders DSL

<u>Grammar</u>:

program → declarationList

<u>Declarations</u>:

declarationList → declaration declarationList | ϵ

declaration → envDecl | agentDecl | behaveDecl | varDecl | fieldAssign | envFunc

envDecl → "*env*" id "{" envBody "}"

agentDecl → "*agent*" id "{" agentBody "}"

behaveDecl → "*behave*" id "{" behaveBody "}"

varDecl → type id " = " expression ";"

fieldAssign → id "." id " = " expression ";"

envFunc → id "." "*reset*" ";" | id "." "*run*" expression ";"

<u>Bodies</u>:

envBody → varDeclList

agentBody → varDeclList

behaveBody → statementList

varDeclList → varDecl varDeclList | ϵ

statementList → statement statementList | ϵ

<u>Statements</u>:

statement → exprStmt | varDecl | repeatStmt | incaseStmt | primFuncStmt

exprStmt → expression ";"

repeatStmt → "*repeat*" "*when*" expression "{" statementList "}" incaseStmt → ”*in*” ”*case*” expression ”{” statementList ”}”*inothercaseStmt*

inothercaseStmt → ”*in*” ”*other*” ”*case*” expression ”{” statementList ”}” inothercaseStmt | ”*otherwise*” ”{” statementList ”}” | ϵ

primFuncStmt → ”*print*” expression ”; ” | ”*move*” expression expression ”; ” | ”*trade*” expression expression expression ”; ” | ”*find*” | ”*random*” expression expression

<u>Expressions</u>:

expression → logicExpr

logicExpr → logicAnd | logicAnd ”*or*” logicExpr

logicAnd → equality | equality ”*and*” logicAnd

equality → comparison | comparison equalityTail

equalityTail → ”$! =$ ” comparison equalityTail | ” $==$ ” comparison equalityTail | ϵ

comparison → term | term comparisonTail

comparisonTail → ” $<$ ” term comparisonTail | ” $<=$ ” term comparisonTail | ” $>=$ ” term comparisonTail | ” $>$ ” term comparisonTail | ϵ

term → factor | factor termTail

termTail → ” $+$ ” factor termTail | ” $-$ ” factor termTail | ϵ

factor → unary | unary factorTail

factorTail → ” $*$ ” unary factorTail | ”/” unary factorTail | ϵ

unary → call | ”!” call | ” $-$ ” call

call → primary | id dotTail

dotTail → ”.” idTail | ϵ

idTail → id dotTail | listFunc dotTail

listFunc → ”*get*” expression | ”*push*” expression | ”*size*” | ”*pop*” | ”*reverse*” primary → ”*true*” | ”*false*” | number | string | ”(” expression ”)”

Lexical Grammar

type → ”*number*” | ”*bool*” | ”*string*” | ”*list*”