Genetic Code Evaluator Report

Architecture

Interface

- Evaluable - node in AbstractSyntaxTree for evaluable expression
    - eval() : int
- Executable - node in AbstractSyntaxTree for executable statement
    - execute()
- token - interface for creating tokenizer for parsing genetic code
    - peek()
    - consume()

Classes

- implements from Evaluable
    - class **Number**
        - number : int
        - eval() : int
            - return the value of the number.
    - class **Identifier**
        - bindings : Map<String, int>
        - variable: String
        - eval() : int
            - return the value of the variable.
    - class **BinaryArith**
        - leftEval, rightEval : Evaluable
        - operation : String
        - eval() : int
            - return the result of the mathematical operation between two expressions.
    - class **SensorExpression**
        - command : String
        - direction : String // If nearby is parsed.
        - eval() : int
            - return the value of the direction obtained by sensor command.
    - class **RandomValue**
        - rand : Random
        - eval() : int
            - return the random value between 0 - 99.

- implements from Executable
    - class **BlockStatement**
        - statements : List<Executable>
        - addStatement()
            - add statement
        - execute()
            - when executed, execute all statements in the block.
    - class **IfStatement**
        - trueStatement : Executable
        - falseStatement : Executable
        - expression : Evaluable
        - execute()
            - when executed, check the condition to choose what statement will be executed next.
    - class **WhileStatement**
        - statement : Executable
        - expression : Evaluable
        - execute()
            - when executed, execute the statement until the value of expression is zero or the loop count reaches 1000 times.
    - class **AssignmentStatement**
        - identifier : String
        - expression : Evaluable
        - bindings : Map<String , int>
        - execute()
            - when executed, it will assign a value into the variable of that Unit.
    - class **MoveCommand**
        - unit : Unit (Virus or ATBD)
        - direction : String
        - execute()
            - when executed , it will make Unit move in the direction specified by the genetic code.
    - class **AttackCommand**
        - unit : Unit (Virus or ATBD)
        - direction : String
        - execute()
            - when executed , it will make Unit shoot in the direction specified by the genetic code.

- implements from token
    - class **Tokenizer**
        - Used in parsing genetic code.
        - src : String
            - the genetic code
        - next : String
            - store the next token of tokenizer
        - initialize(geneticCode : String)
            - reset the tokenizer and initialize the source genetic code
        - computeNext()
            - compute the next token
        - peek() : String
            - return the next token.
        - consume() : String
            - remove the next token and return it.
        - peek(token : String) : boolean
            - check if the next token is equal to the desired string. If not, throw an error.
        - consume(token : String)
            - check if the next token is equal to the desired string. If not, throw an error. Otherwise, remove the token and return it.
        - hasNext() : boolean
            - check if there is a token left or not.
        - isSpace(c : char) : boolean
            - check if a character is space.
        - isAlphaNumeric(c : char) : boolean
            - check if a character is an alphabet, number or underscore.
- class **GeneticEvaluator** - generate program that store in the unit
    - tkz : **Tokenizer**
    - unit : Unit
    - bindings : Map<String,int> - bindings of this unit.
    - instance : static GeneticEvaluator - singleton design
    - generate(): Executable
        - generate the program by calling parseProgram() to parse the genetic code of the unit and return the program to the unit itself.
    - grammar parser method
        - parseProgram() : Executable
        - parseStatement() : Executable
        - parseCommand() : Executable
        - parseAssignmentStatement() : Executable
        - parseActionCommand() : Executable
        - parseMoveCommand() : Executable
        - parseAttackCommand() : Executable
        - parseBlockStatement() : Executable
        - parseIfStatement() : Executable
        - parseWhileStatement() : Executable
        - parseExpression() : Evaluable
        - parseTerm() : Evaluable

        - parseFactor() : Evaluable
        - parsePower() : Evaluable
        - parseRandom() : Evaluable
        - parseSensorExpression() : Evaluable
    - isNumber(): boolean - to check if the token is the number

What did we learn?

- good management and design leads to simple implementation
- many exceptions can quickly discover a bug

Testing

How did you test your parser and evaluator?

- Use JUnit 5 to test most of test cases and coverage input space

Describe test cases you used for testing

- Tokenizer Testing
    - unidentified token type
    - the token specified cannot be found or consumed
- Evaluator Testing
    - Wrong format/grammar of genetic coding
    - Missing Number, Identifier, SensorExpression or RandomValue
    - Negative number in genetic coding
    - Use reserved word as identifier
    - Unidentified direction word
    - Math error from Expression

What did your group learn from testing?

- Many tests ensure a sustainable program.

Work plan

explain the role of member(s) in your group responsible for this part of the project

- All classes and interfaces of parser, tokenizer and evaluator are designed and implemented by Pawaret Dilokwuttisit.
- did your group change the division of work? if so, why, what was the change, and how did you end up with that change?
    - Yes, Ratchanon does not understand Spring Framework and the front end. Ratchanon swapped responsibilities with Pawaret (between the front end and game class). After changing, it's soooo nice.
- New work plan:
    - Parser and Evaluator - Done
    - Frontend (React), Backend (API) - Pawaret Dilokwuttisit
    - Backend (Game Classes) - Ratchanon Niwat , Saranphot Chesrichai

what did your group learn from the process of dividing up the work, including changes along the way (if any)?

- Dividing up the work leads to a smooth and efficient flow of work.

Known problems (optional)

- detail any known problems with your specification, design, or implementation (if not already mentioned before)
    - We haven’t created an API before so it is quite unpredictable and might take a lot of time. (▰︶︹︺▰)
    - Our React skill is not so strong
    - At this point, do you believe you will be able to finish the entire project on time? If not, why not, and when do you plan to have the entire project completed, and why?
        - We are not so positive about this, since we have so many things to do about the Spring Framework , but we will still try to finish the project on time. If not, an extra one week is enough for us.