CARIN

Design Overview Document

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Model-View-Controller

Model:

- timeUnit
- Unit
    - Type
    - Unit stats
- Virus
- Antibody
- Field
- Objective
- Currency

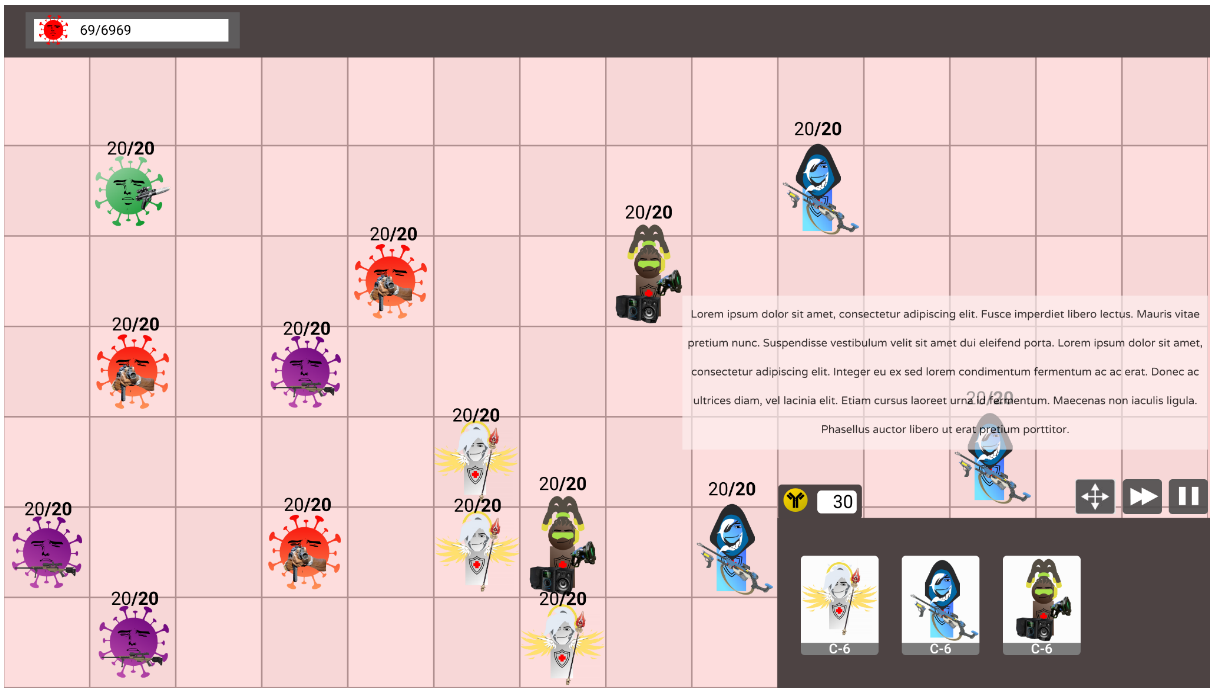
View:

- Button
    - Move
    - Speed
    - Pause
    - Cancel
- Virus & Antibody
- Antibody Tiles
- Cell Tiles
- Currency Tabs
- Objective Bar
- Unit Description
- HP Gauge

Controller:

- Button
- Mouse Status
- Keyboard Binding

Display Outline :



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce imperdiet libero lectus. Mauris vitae pretium nunc. Suspendisse vestibulum velit sit amet dui eleifend porta. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu ex sed lorem condimentum fermentum ac ac erat. Donec ac ultrices diam, vel lacinia elit. Etiam cursus laoreet urna id fermentum. Maecenas non iaculis ligula. Phasellus auctor libero ut erat pretium porttitor.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Design:

- Architecture:

Game Classes:

- implements from *Unit*
    - class ***Virus***
        - **Rep Invariants:**
            - **position will not exceed the dimension of body (m,n)**
            - **every stat is always an positive int**
            - **genetic code must be appropriate grammar format**
        - bindings : Map<String,int>
            - unit's local variables
        - position : Pair<int,int>
        - stats
            - health : int
            - attack : int
        - cost : int
            - unit's cost
        - geneticCode : String
            - unit's genetic code
        - program : *Executable*
            - this is where this unit execute the genetic code by calling execute()
        - move({x,y} : Pair<int,int>) **Observer**
            - notify ***Game*** that this Unit want to move
        - shoot(direction : String)
        - destruct() **Observer**
            - notify ***Game*** that this Unit dies
            - update ATBD health //health restoring
        - setter() stats modifying method
        - takingDamage(enemy : ***ATBD***) - modify health and record attacker

Currently, we design 3 types of virus which each of them have different genetic codes and stats this maybe change in the future for game balance possible extension: defensive stat *still need further testing

- class ***ATBD***
    - **Rep Invariants:**
        - **position will not exceed the dimension of body (m,n)**
        - **Every stat is always an positive int**

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

        - **Genetic code must be appropriate grammar format**
    - bindings : Map<String,int>
        - unit's local variables
    - position : Pair<int,int>
    - stats
        - health : int
        - attack : int
    - cost : int
        - unit's cost
    - geneticCode : String
        - unit's genetic code
    - program : *Executable*
        - this is where this unit execute the genetic code by calling execute()
    - move({x,y} : Pair<int,int>) **Observer**
        - notify ***Game*** that this Unit want to move
    - shoot(direction : String)
    - destruct() **Observer**
        - notify ***Game*** that this Unit dies
        - notify ***Game*** to spawn new virus that has the same genetic code of ***ATBD*** that it kills
    - setter() - stats modifying method
    - takingDamage(enemy : ***Virus***) - modify health and record attacker

3 types of ATBD. different costs and stats; players need to utilize them according to the circumstances.

*extensions are expected to be added after the game is completed.

- class ***Shop*** : **Singleton** class to manage ATBD purchasing function for Frontend displaying. If currency reaches a tower's cost its tile glows and becomes clickable.
    - **Rep Invariants : Currency is always positive**
    - currency : int
    - Map<ATBD,Boolean> true if currency >= ATBD.cost this field will affect displaying(frontend)
    - updateBool():for loop of the Map to update the boolean value in each time unit This method will be called by ***Game*** class

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- class ***Game*** : **Singleton**
    - **Rep Invariants :**
        - **spawn's, remove's and move's parameters will not exceed dimension of body (m,n)**
        - **order must not have a duplicated Unit.**
        - **objective Pair<x,y> x always >= 0 | y always >= 1**
    - geneticEvaluator : **GeneticEvaluator**
        - evaluator for genetic code.
    - shop **: Shop**
    - state : int
        - the current state of the game
    - Field : **Unit**[m][n]
        - contains units in the field.
    - order : List<**Unit**>
        - evaluation order
    - config file variable :
        - m,n : int
            - the dimension of field (m - rows , n - columns)
        - virusSpawnRate : double
            - probability of virus spawned
            - value between 0 - 1
        - baseCredits : int
            - the antibody credits you get in the beginning of the game.
        - placementCost : int
            - the cost of antibody placement.
        - baseVirusHP : int
            - maximum HP of viruses.
        - baseATBDHP : int
            - maximum HP of antibodies.
        - baseVirusDamage : int
            - virus attack damage.
        - baseVirusAttackGain : int
            - the amount of healing when a virus attacks an antibody.
        - baseATBDDamage : int
            - antibody attack damage.
        - baseATBDKillGain : int
            - the amount of healing when an antibody kills a virus.
        - moveCost : int
            - antibody move cost.
    - objective : Pair<int, int>
        - number of remaining viruses to be eliminated and maximum amount of viruses to be spawned.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- virusLimit : int
    - the maximum number of viruses in the field.
- config()
    - method used in parsing configuration file
- spawn(unit : *Unit* , {x,y} : Pair<int, int>)
    - spawn a unit in a specified location in field
- remove({x,y} : Pair<int, int>)
    - remove a unit from specified location in field
- move(unit : *Unit* , {x,y} : Pair<int, int>)
    - move a unit to another specified location
- sensorClosestVirus(unit : *Unit*) : int
    - find the closest virus from this unit.
- sensorClosestATBD(unit : *Unit*) : int
    - find the closest antibody from this unit.
- sensorNearby(unit : *Unit* , direction : String) : int
    - find the closest unit in the specified direction from this unit.
- update() // Run everything in one time unit
    - spawnVirus()
        - spawning virus in the field.
    - runGeneticCode()
        - evaluate the genetic code of all units in order.
    - updateShop()
        - update the purchasable tile in the shop.
    - updateField()
        - update all the variables in the field.
    - updateObjective()
        - update the remaining objective
- main()
    - use in running games

Parser Classes:

- implements from *Evaluable*
    - class ***Number***
        - **Rep invariants:**
            - **The number must be a nonnegative integer.**
        - number : int
        - eval() : int
            - Return the value of the number.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- class **Identifier**
  - **Rep invariants:**
    - **The identifier's name must not be a reserved word.**
  - binding : Map<String, int>
  - identifier : String
  - eval() : int
    - Return the value of the variable.
- class *BinaryArith*
  - **Rep invariants:**
    - **Division or Modulo by zero is not allowed**
  - leftEval : *Evaluable*
  - 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 command.
- class ***RandomValue***
  - eval() : int
    - Return the random value between 0 - 99.

- implements from *Executable*
  - class ***BlockStatement***
    - **Rep invariants:**
      - **statements can have zero or more *Executable***
    - statements : List<*Executable*>
    - 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.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- 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***
    - direction : String
    - execute()
        - When executed , it will make Unit move in the direction specified by the genetic code.
- class ***AttackCommand***
    - direction : String
    - execute()
        - When executed , it will make Unit shoot in the direction specified by the genetic code.

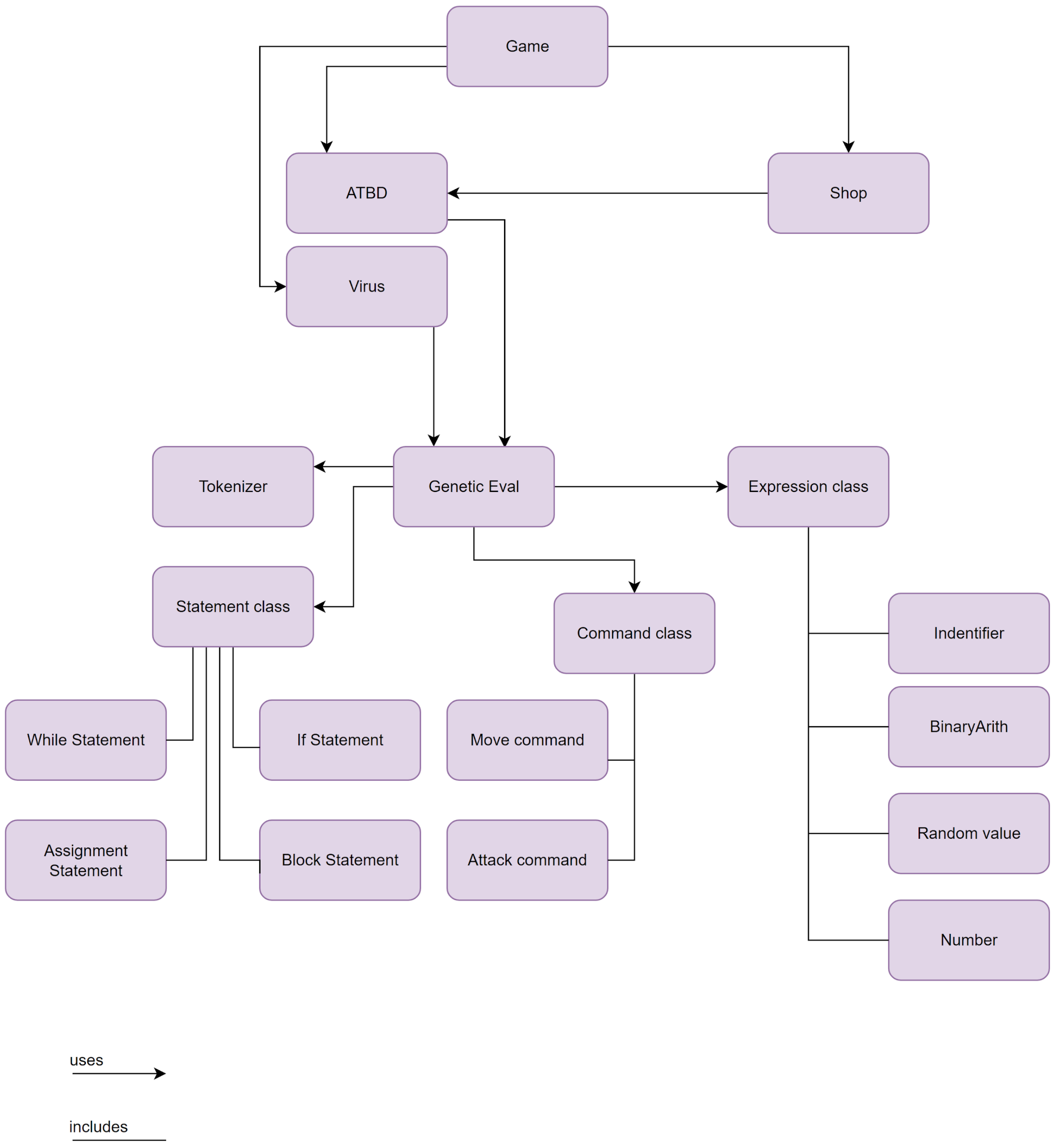
- implements from *tokenizer*
    - class ***Tokenizer***
        - **Rep invariants:**
            - **genetic code must only have alphanumeric character and 11 special characters [+ , - , * , / , % , ^ , (,) , { , } , =]**
        - Used in parsing genetic code.
        - src : String
            - the genetic code
        - next : String
            - store the next token of tokenizer
        - peek()
            - Return the next token.
        - consume()
            - Remove the next token and return it.
        - peek(token : String)
            - 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.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- class ***GeneticEvaluator*** **Singleton**
    - **Rep invariants:**
        - **genetic code must be appropriate grammar format**
    - tkz : ***Tokenizer***
    - unit : *Unit*
    - bindings : Map<String,int>
        - to store bindings of this unit.
    - geneticEvaluator : *Executable*
        - This is where you execute genetic code from a Unit.
    - grammar parser method
        - parseProgram() : *Executable*
        - parseStatement() : *Executable*
        - parseCommand() : *Executable*
        - parseAssignmentStatement() : *Executable*
        - parseActionCommand() : *Executable*
        - parseMoveCommand() : *Executable*
        - parseAttackCommand() : *Executable*
        - parseDirection() : *Evaluable*
        - parseBlockStatement() : *Executable*
        - parseIfStatement() : *Executable*
        - parseWhileStatement() : *Executable*
        - parseExpression() : *Evaluable*
        - parseTerm() : *Evaluable*
        - parseFactor() : *Evaluable*
        - parsePower() : *Evaluable*
        - parseSensorExpression() : Evaluable

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Dependency Graph:



Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Key Interfaces:

- Unit (Virus and Antibody)
    - move()
    - shoot()
    - destruct()
    - setterMethod method to modify private field such as setHP()
- token - interface for creating tokenizer for parsing genetic code
    - peek()
    - consume()
- Evaluable - Node in AbstractSyntaxTree for evaluable expression
    - eval() : int
- Executable - Node in AbstractSyntaxTree for executable statement
    - execute()

Design Pattern:

Singleton - use with object that only need 1 instance
=> class Shop , class Game , class GeneticEvaluator

Observer - use with object's method that need to communicate
move(),destruct() in Unit

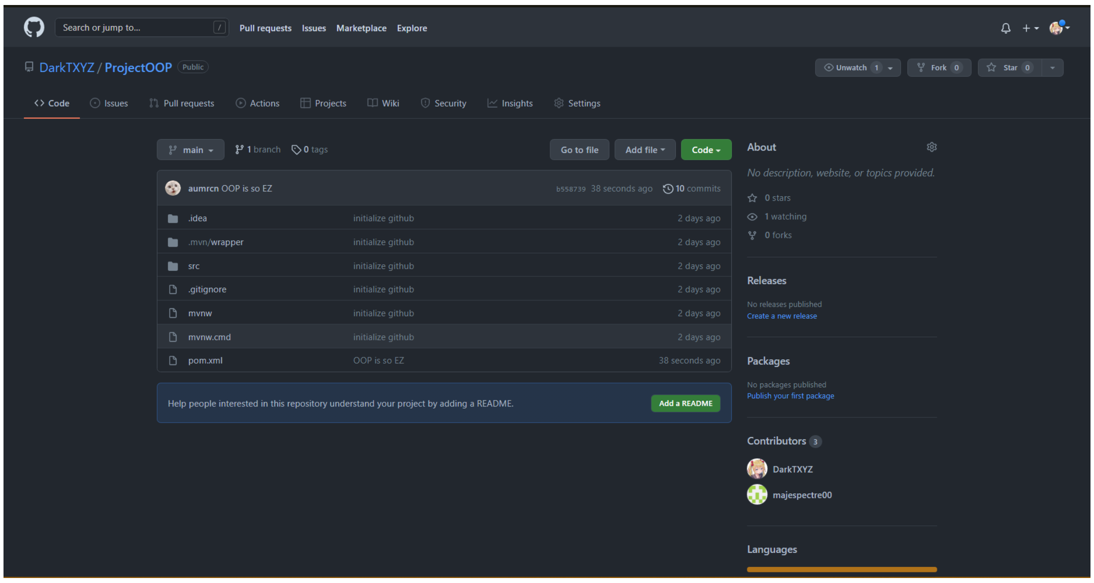
State - button in frontend since they have state for example toggling

- ## Code Design:
    - Data Structures
        - Pair<x,y> :to store any two data that need to use at the same time
        - List<> : to store data that need to be store/use with order which is the ***Unit***
        - Array[][] : to store ***Unit***; representing their position in the field
        - Map<> : to store the word mapped to value , used to get the value of variable in genetic code
    - Design Goal and Trade-off
        - Our team wants to make the game "simple" so the implementation will be quite straightforward and easy to understand. But, the implementation and runtime may be inefficient in some cases.

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- **Tools:**
    - Display Design : Figma
    - Backend : Java(VScode, IntelliJ)
    - Frontend : Spring Framework , HTML , React
    - Communication : Discord

- **Github Screenshot:**
    - https://github.com/DarkTXYZ/ProjectOOP



DarkTXYZ / ProjectOOP Public

1 branch 0 tags

aumrcn OOP is so EZ b558739 38 seconds ago 10 commits

.idea	initialize github	2 days ago
.mvn/wrapper	initialize github	2 days ago
src	initialize github	2 days ago
.gitignore	initialize github	2 days ago
mvnw	initialize github	2 days ago
mvnw.cmd	initialize github	2 days ago
pom.xml	OOP is so EZ	38 seconds ago

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

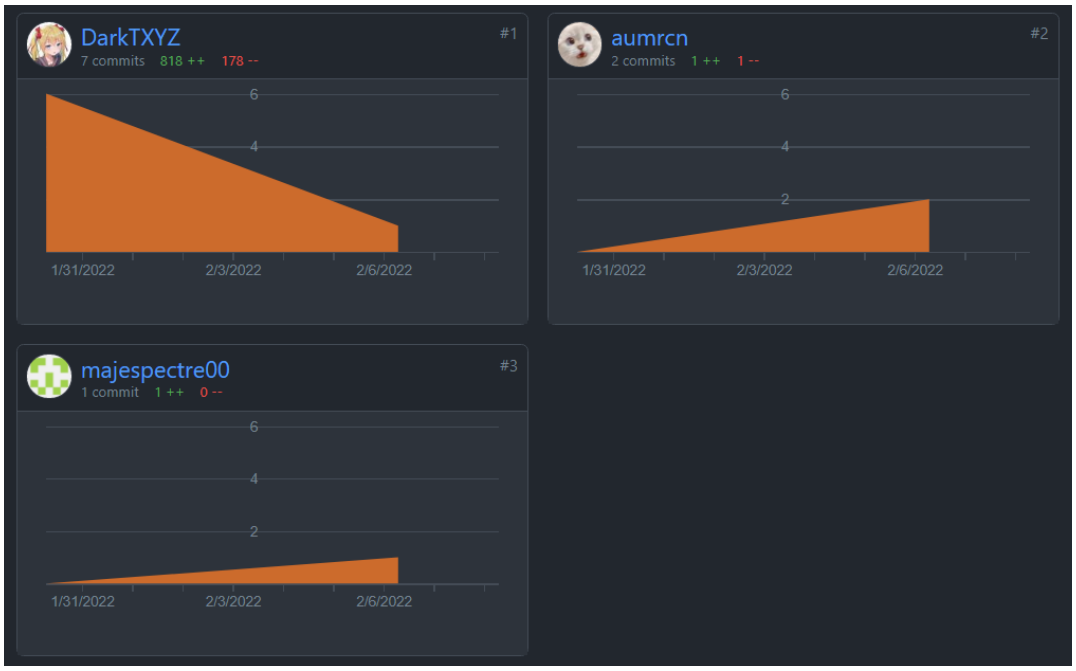
Packages

No packages published
Publish your first package

Contributors 3

DarkTXYZ
majespectre00

Languages



Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

- **Testing**
  - Will your test cases have enough coverage of the input space?
    - testing ordinary user inputs. input(function calling) which come from frontend player’s controller we provide such as moving, purchasing ATBD, choosing cell
    - testing from project specification

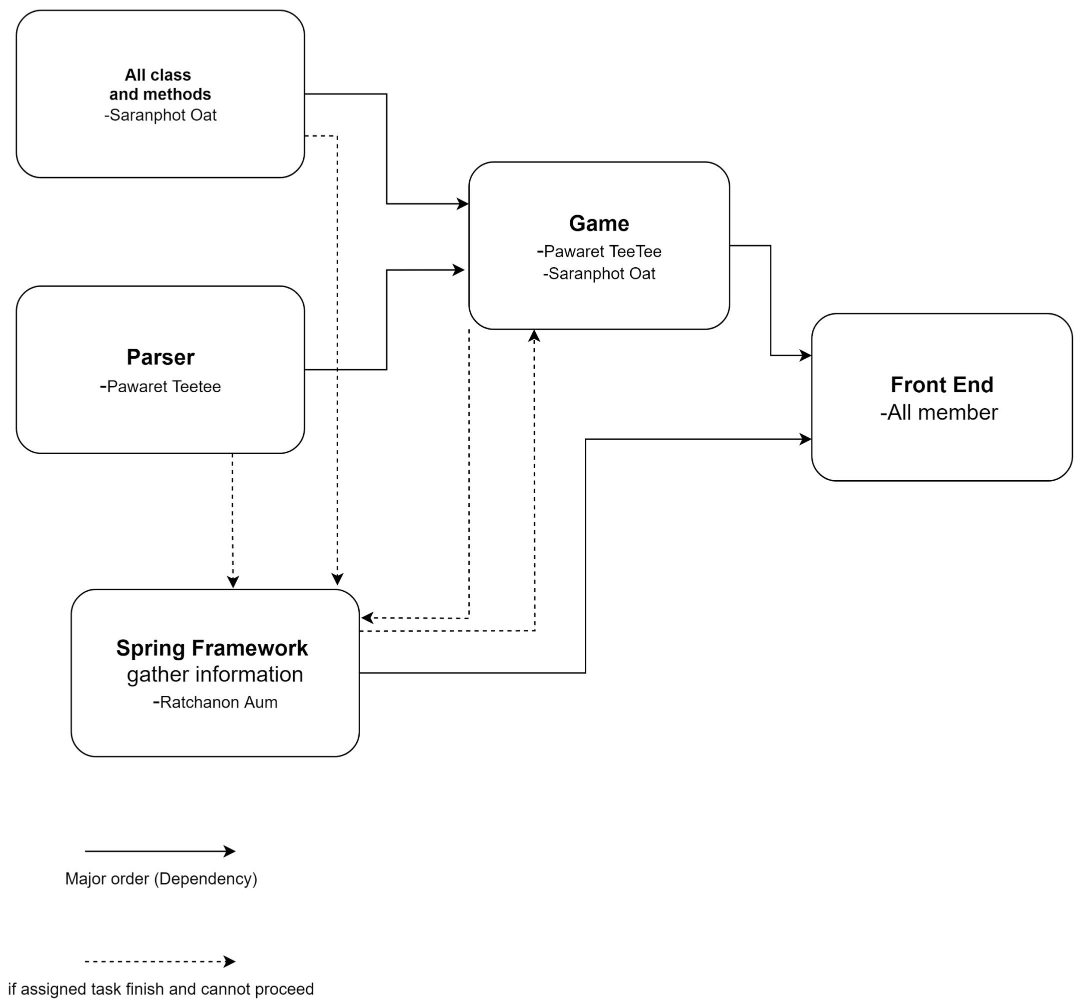
  - Will your test cases have enough coverage of your code?
    - test all written functions and their branches
    - mostly testing parser and unusual inputs in functions which possibly happen by ***Game*** class
    - Evaluator Testing
      - Input (can detect error and throws an exception)
        - Wrong format/grammar of genetic coding
        - Negative number in genetic coding
        - Use reserved word as identifier
      - Math error from Expression

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Work Plan:

มีการแบ่งงานให้ 2 ส่วนคือ งาน front-end กับ back-end

- (1 คน) ส่วน Front-end จะไปศึกษาเรื่อง Spring Framework และการใช้งาน
- (2 คน) คนที่ทำงานส่วน Back-end จะแบ่งอีก 2 ส่วน คือ เขียน Evaluator กับ เขียน Class ที่ใช้ในเกม ซึ่งหากมีคนใดคนหนึ่งทำเสร็จแล้ว จะต้องไปช่วยฝั่ง Front-end ก่อน
- เส้นตรง คือ dependency ของงาน คือต้องทำงานที่หางก่อนจึงทำงานที่หัวลูกศรได้
- เส้นประคือในกรณีที่งานหลักของแต่ละคนเสร็จแล้ว อาจจะไปช่วยเหลือตามเส้นประ



Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.

Known problem:

- Frontend Development Problems
    - ไม่แน่ใจ interactions ระหว่าง frontend กับ backend เบื้องต้นเข้าใจว่าเราให้ backend จัดการ process ของเกมทั้งหมด แล้ว frontend ทำการ display ข้อมูลเกมออกมาและรับ input มาให้ backend ทำการ process แต่ frontend จะส่ง input ให้ backend ยังไงนั้นยังไม่ชัดเจน
- Work Time
    - ในกลุ่มของเราได้มีการวางแผนที่จะทำงานให้ทันกำหนดส่ง ดังนั้นหากมีการขยายระยะเวลาการทำงาน จะยังคงทำตามแผนเดิมอยู่ แต่เวลาในส่วนที่เพิ่มมานั้น จะใช้สำหรับตรวจสอบงาน
    - แต่ถ้าเกิดปัญหาขึ้นมาระหว่างการทำงานแล้วทำให้งานล่าช้า เวลาที่เพิ่มมานั้นจะเป็นผลประโยชน์กับกลุ่มของเราเป็นอย่างมาก

Comments:

- How much time did you spend on the design?
    - 20.00 - 01.00 daily
    - ~20hrs
- What advice should we have given you before you started?
    - advices about spring framework or a head start
- What was surprising about the design?
    - At first, we thought the scope of design was small and narrow. But when we design a class, we know that the scope is bigger than we can imagine.
- What was hard about the design?
    - the design 😢 ـิ.ـั

Italic and Bold word means class.
Italic word means interface.
Red and Bold word means design pattern.