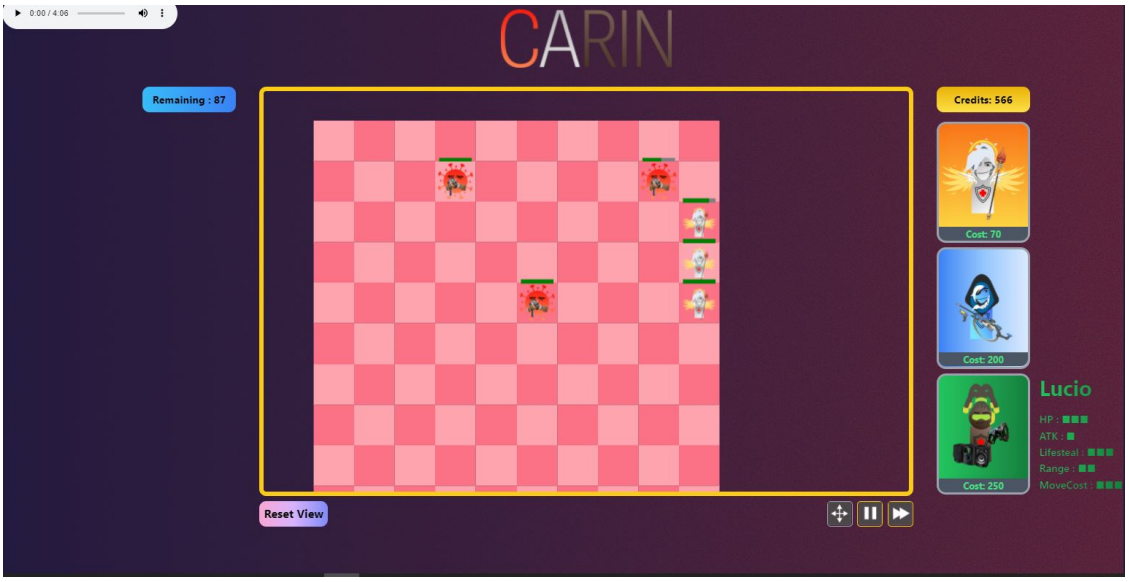
CARIN

Design Overview Document

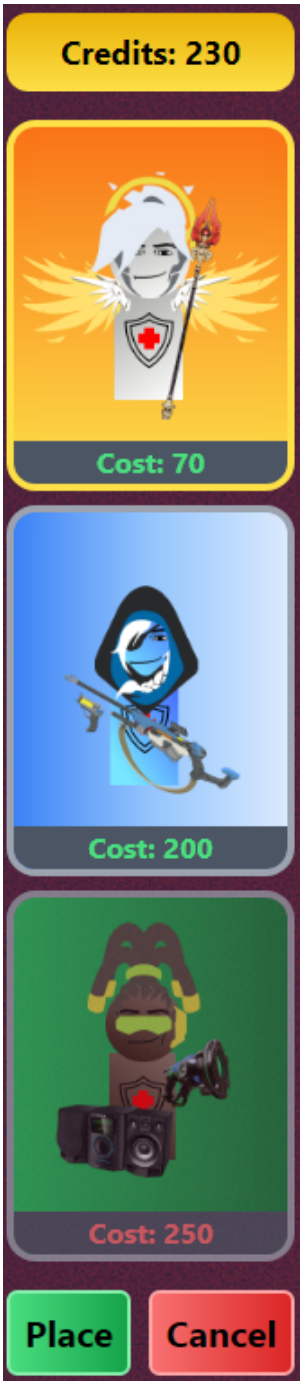
Gameplay

- Screenshot :



- Objective : กำจัดไวรัสให้ครบจำนวนที่กำหนด
- How to play :
    - Summary : สร้าง Antibody เพื่อกำจัด Virus ให้ครบตามจำนวน โดยในทุกๆครั้งที่ Virus ตายผู้เล่นจะได้รับ Antibody Credit เพื่อใช้ในการซื้อ Antibody เพิ่มเติม เงื่อนไขที่ผู้เล่นจะแพ้คือ
        - 1. หาก Credit ของผู้เล่นหมด และ Objective ยังเหลืออยู่
        - 2. หาก Virus สามารถยึดครอง Field ทั้งหมด

    - เลือก Antibody ที่ต้องการสร้างจากบริเวณทางขวาของจอ จากนั้น เลือกช่องที่ต้องการ และกด Place เพื่อวาง หรือ Cancel เพื่อยกเลิก

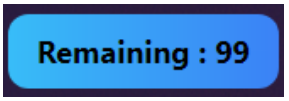


← แสดง Antibody Credits ที่มีอยู่

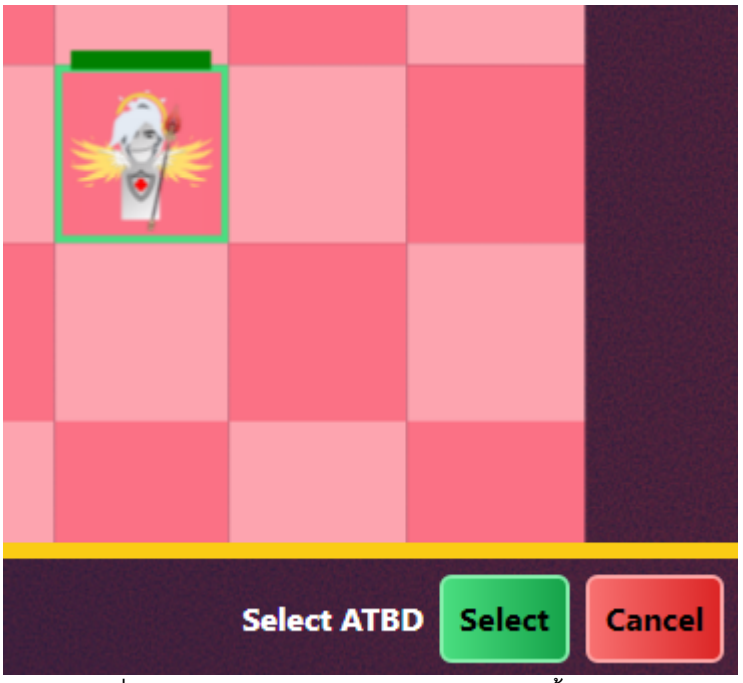
← แสดงถึงราคาของ Anitbody แต่ละตัว หากขึ้นสีเขียวหมายความว่าสามารถซื้อได้

← และจะเป็นสีแดงหากซื้อไม่ได้

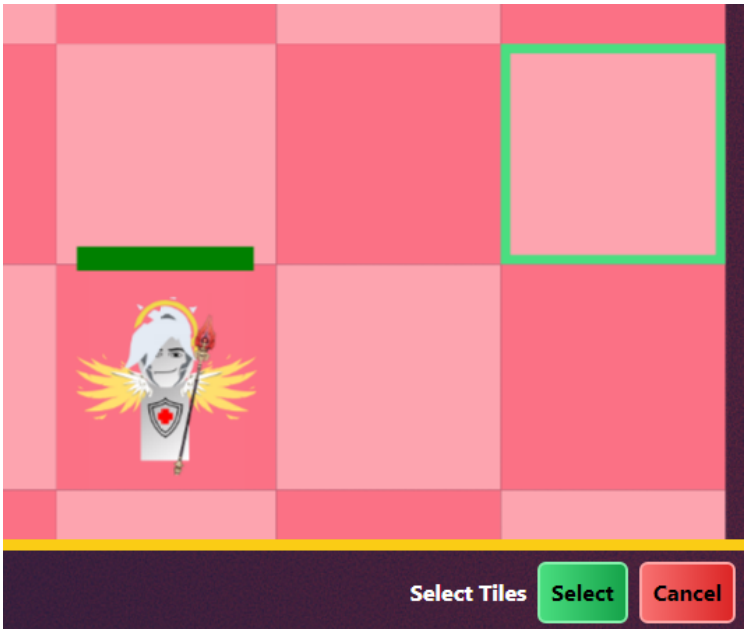
← กดปุ่ม Place เพื่อวาง และกดปุ่ม Cancel เพื่อยกเลิก

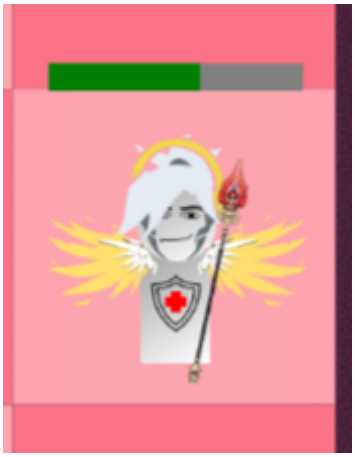
- การแสดงผลของ Objective :
    - แสดงจำนวนของไวรัสที่เหลือที่ต้องกำจัด : 

- การใช้ปุ่ม Move
    - หน้าตาของปุ่ม  :
    - กดเพื่อย้าย Anitbody ที่เลือกไปยังช่องที่ผู้เล่นต้องการ
    - ควรใช้ปุ่มนี้เพื่อที่จะประหยัด **Antibody Credit**
    - เมื่อกดปุ่มแล้วให้ทำการเลือกช่องที่มี Antibody ยืนอยู่ จากนั้น กดปุ่ม Select เพื่อยืนยันการเลือก หรือ กดปุ่ม Cancel เพื่อยกเลิก



    - เลือกช่องที่ต้องการจะย้าย Antibody ไป จากนั้น กดปุ่ม Select เพื่อยืนยันการเลือก หรือ กดปุ่ม Cancel เพื่อยกเลิก

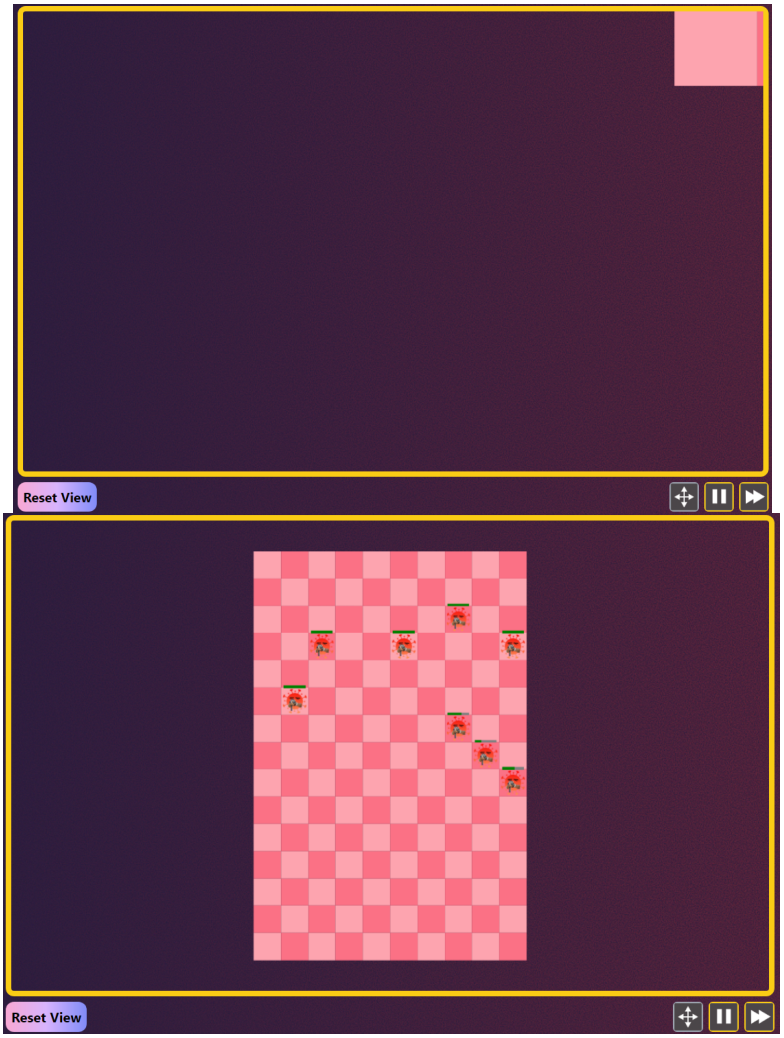


- จากนั้น Antibody จะถูกย้ายไปช่องที่เลือก และจะมี HP ที่ลดลงไปตามค่า Move cost
- 

- การใช้ปุ่ม Pause
  - หน้าตาปุ่ม  :
  - กดเพื่อหยุดการทำงานของเกมชั่วคราว
  - ควรใช้ปุ่มนี้เพื่อการวางแผน
  - หากกำลัง Pause อยู่ขอบของปุ่มจะกลายเป็นสีเหลือง  :
- การใช้ปุ่ม 2x Speed
  - หน้าตาปุ่ม  :
  - กดเพื่อเพิ่มความเร็วของเกมเป็น 2 เท่าจากเดิม 1 เท่า โดยเป็นปุ่มแบบ Toggle
  - หากเกมกำลังมีความเร็ว 2 เท่าอยู่ขอบของปุ่มจะกลายเป็นสีเหลือง : 

- การใช้ปุ่ม Reset View
    - หน้าตาปุ่ม 

:
    - ใช้เพื่อ Reset ตำแหน่งของตารางให้กลับมาอยู่ตรงกลางของหน้าจอ (เนื่องจากเป็น bug ของทาง library การซูม)
    - ก่อนกดปุ่ม :
    - หลังกดปุ่ม :



- Special Specifications
    - No Friendly Fire
    - Player’s genetic code can be provided to games by put it in the geneticcodeInput.in
    - Players can modify and config game values by adjusting in the configfile.in and config values can be negative in some circumstances

Architecture

ri- stands for rep invariant

Key Interfaces:

- Evaluable
  - eval()
- Executable
  - execute()
- token
  - peek()
  - consume()
- Unit
  - void move(String direction);
  - void shoot(String direction);
  - void destruct();
  - void takingDamage(Unit attacker);
  - void execute() throws DeadException;
  - void setProgram(Executable program);
  - void attack(Unit a);
  - int getAtk();
  - int getHp();
  - int getMaxHp();
  - int getLifeSteal();
  - int getCost();
  - int getAttackRange();
  - String getGene();
  - Map<String,Integer> getBindings();
  - Pair<Integer,Integer> getPosition();
  - void setPos(Pair<Integer,Integer> pos);
  - void setHP(int mod);
  - void setAttack(int mod);
  - public void configMod(int cAtk, int cLs, int cHp, int cost, int cmoveCost);
  - public void setDf(int datk, int dls,int dhp, int dcost,int datkrange, int dmovecost);

Classes:

Game Classes:

- Abstract Class Unitimpl any Units' method and field which has the same logic (probably getter setter move) will be impl here.
  - Field :
    - *int* Hp; ri: cannot be negative
      - Unit's health

        - *int* maxHp; ri: cannot be negative
            - Unit's maximum health(hp with not exceed maxhp)
        - *int* Atk; ri: cannot be negative
            - Unit's potential to inflict damage
        - *int* lifeSteal; ri: cannot be negative
            - for Virus it is health restoration when attacking
            - for ATBD it is Kill gain
        - *int* cost; ri: cannot be negative
            - storing cost of Unit
            - For Virus it is a credit drop.
        - *int* moveCost; ri: cannot be negative
            - move penalty of Unit
        - *int* attackRange; ri: >0
            - maximum shoot range
            - use in shoot()
        - int skin ri:1-6 only
            - for frontend display.
        - Boolean isDead;
            - storing Dead status of Unit
        - String geneticCode;
            - the geneticcode
        - Unit previousAttacker;
            - storing the last attacker to be use in destruct()'s logic
        - Executable program; is from parsing genetic code
            - run every time unit
        - Map<String,Integer> bindings;
        - Pair<Integer,Integer> position; ri: cannot be <0 cannot exceed m,n
    - Method :
        - *void* execute() throws DeadException
            - execute program
            - if unit is dead will throw DeadException (for Game's loop to remove it from Iterating)
                - this "if" debug concurrent Exception in ***Game*** class
        - *void* attack(Unit a)
            - call a.takingDamage
        - *void* abstract destruct()
        - *void* move(String direction)
            - array logic
            - create a param for gmove
        - *void* gmove(Pair<Integer, Integer> destination)

        - call Game.gmove() to move the unit in field[][]
    - *void* abstract shoot(String direction)
    - *void* takingDamage(Unit attacker)
        - mod hp and set isDead if hp reach 0
    - *void* configMod(*int* cAtk, *int* cLs, *int* cHp, *int* ccost, *int* cmoveCost)
        - mod stats read from config file
    - public void setDf(int datk, int dls,int dhp, int dcost,int datkrange, int dmovecost){
        - set our game default atbd and virus stats
    - getter method
        - *int* getHp()
        - *int* getAtk()
        - *int* getMaxHp()
        - *int* getLifeSteal()
        - *int* getCost()
        - *int* getAttackRange()
        - Pair<Integer,Integer> getPosition()
        - String getGene()
        - Map<String, Integer> getBindings()
    - setter method
        - *void* setAttack(*int* mod)
        - *void* setPos(Pair<Integer,Integer> pos)
        - *void* setHP(*int* mod)
        - public void setProgram(Executable program)
- implements from *Unit*
    - class ***Virus***
        - set of contructor
            - construct with all stat
                - public Virus(int atk, int lifeSteal, int hp, String gene,int attackRange)
            - construct from created Unit
                - Virus(Unit template)
        - shoot(direction : String)
            - call senseNearby of ***Game*** class for target
            - will only shoot ATBD
        - destruct() **Observer**
            - notify ***Game*** that this Unit dies by calling Game.destroyVirus
            - update the attacker's health (which can only be ATBD) by calling its setter method
    - class ***ATBD***
        - set of contructor
            - similar to Virus's constructor

        - shoot(direction : String)
            - similar to Virus's shoot method but only shoot Virus
        - destruct() **Observer**
            - notify ***Game*** that this Unit dies by Calling ***Game***.destroyATBD

- 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.
    - int currency : spend to buy atbd ri:cannot be <0
    - int maxCurrency : maximum credits player can hold ri: cannot be <0
    - Map<Integer,Boolean> status
        - this field will be sent to API
        - Integer is for the cost of all ATBDs
        - Boolean will be true if currency >= cost
    - updateStatus():for loop of the Map to update the boolean value in each time unit This method will be called by ***Game*** class
    - Shop(int[] cost)
        - Constructor (singleton)
        - param cost to init the status field mention above
        - will be called in ***Game***.Initialize()
    - static getInstance(int[] cost)
        - Singleton thingy
    - static updateCost(int[] cost)
        - method to update status field after if atbds cost have been changed
        - will be called in ***Game***.Initialize()
    - setdf(int default):set default currency of our game
    - getcostList , getStatus: getter method
- class ***Game*** : **Singleton**
    - **Fields :**
        - public static Game *instance*;
            - game instance
        - default unit
            - static Unit *gangster* = new Virus();
            - static Unit *pistolDude* = new Virus();
            - static Unit *sniper* = new Virus();
            - static Unit[] *viruses* = {*gangster*, *pistolDude*, *sniper*};
            - static Unit *Merci* = new ATBD_();
            - static Unit *Ana* = new ATBD_();

    - static Unit *Lucio* = new ATBD_();
    - static Unit[] *Atbds* = {*Merci*, *Ana*, *Lucio*};
    - static int[] *cost* = new int[3];
    - static int *pause*;
    - static int *speed*;
- protected final String inFile;
    - config file
- protected final String geneinFile;
    - player input gene file
- protected final String geneDefault;
    - default gene we provided
- protected String[] geneATBD = new String[3]; // init genetic code for each ATBD
    - store read atbds' gene
- protected String[] geneVirus = new String[3]; // init genetic code for each Virus
    - store read viruses' gene
- protected String[] defaultGeneATBD = new String[3];
    - store default gene
- protected String[] defaultGeneVirus = new String[3];
    - store default gene
- Comparator<String> comp
    - comparator for empty tile
- Queue<Pair<Unit, Pair<Integer, Integer>>> deadList
    - store dead atbd before add to order(prevent concurrent in gene loop)
- int lowestcost
    - store lowest atbd cost for config()
- int highestcost
    - store highest atbd cost for shop logic
- boolean how2Play = false;
    - for frontend state display
- boolean waitingRestart = false;
    - for frontend state display
- private double spawnCount;
    - for spawn mechanic
- private double virusSpawnRate; ri:cannot be <=0 , cannot exceed >1
    - from configfile
    - for spawn mechanic
- private SortedSet<String> emptySlot;
    - for ramdomTile() to spawn virus on empty slot
- private List<Unit> order;
    - Order of unit in field[][]

        - private List<Unit> virusOrder;
            - Order of virus only
        - private List<Unit> atbdOrder;
            - Order of atbd only
        - private Objective gObjective;
            - game objective(win condition)
        - private int virusLimit;
            - the maximum amount of viruses that can be spawn
        - private int limitCount;
            - count how many virus has been spawned
        - private Shop shop;
            - shop
        - private int m, n; ri:cannot be<1, both cannot be 1 at the same time(no 1x1 field)
            - field size
        - private Unit[][] field;
            - game field
            - front end display

    - Method :
        - ***public void*** initObjective(***int*** maxElim);
            - set initial objective of the game
        - ***public void*** notifyReachElim();
            - notify the player if the player wins or loses.
        - ***public*** Pair<Integer,Integer> randomTile();
            - return a random empty tile in the field.
        - public void updateDeadList();
            - add virus to order(prevent concerrent in genetic loop)
        - ***public void*** add(Unit unit, Pair<Integer,Integer> position)
            - root method of adding unit
            - add units into the list of units and field[][].
        - ***public void*** addATBD(Unit a, Pair<Integer,Integer> position)
            - add antibodies to the input position and in the list of antibodies if that position is empty.
            - call static add()
        - ***public void*** addVirus(Unit v, Pair<Integer,Integer> position)

    - add virus to the input position and in the list of viruses if that position is empty.
    - call static add()
- ***public void*** remove(Pair<Integer,Integer> position)
    - root method of removing unit
    - remove the unit that stores in that position of field[][].
- ***public void*** moveATBD(Unit u,Pair<Integer,Integer> destination)
    - will be called by getInput() and input come from frontend user's control
    - move antibodies to the destination, the player pays that atbd's health for this method.
- ***public void*** move (Unit u, Pair<Integer,Integer> destination)
    - root method of moving unit
    - move the unit to the destination without cost.
- ***public void*** destroyATBD(Unit unit, Unit spawn)
    - will be called by method in ***ATBD*** class
    - destroy the input antibody and spawn the new virus that killed the input antibody.
    - call static remove()
- ***public void*** destroyVirus(Unit unit)
    - will be called by method in ***Virus*** class
    - destroy the input virus.
    - update game objective and mod shop currency
    - call static remove()
- ***public void*** visualize()
    - visualize the field in the terminal.
- ***public int*** senseClosestEnemy(Unit u,int type)
    - find closest Enemy according to param type
- ***public int*** senseClosestVirus(Unit u)
    - call senseClosestEnemy (atbd type)
- ***public int*** senseClosestATBD(Unit u)
    - call senseClosestEnemy (virus type)
- ***public int*** senseNearby(Unit u, String direction)
    - find the closest unit from the input direction of the input unit.
- ***public void*** gShoot(Pair<Integer,Integer> pos, Unit u)
    - unit u attack the unit in the input position.
    - calling unit in target position.takingDamage()

        - this is because we only point unit by position in field[][]
    - public Unit createNewVirus(***int*** n)
        - create a new virus type n (there are 3 different types of virus).
    - public Unit createNewATBD(***int*** n)
        - create a new antibody type n (there are 3 different types of antibodies).
    - ***public void*** Initialize()
        - initialize game field with default value
        - add config value to those field
    - ***public void*** Update()
        - update the game(genetic loop, spawn,api communication) while the game is not over.
    - public void getInput()
        - get inputs from API and execute them
    - public geneticReader()
        - read player's input genetic code
    - public geneticDefaultReader
        - read dev's provided default genetic code
    - public void config()
        - read config file

Parser Class:

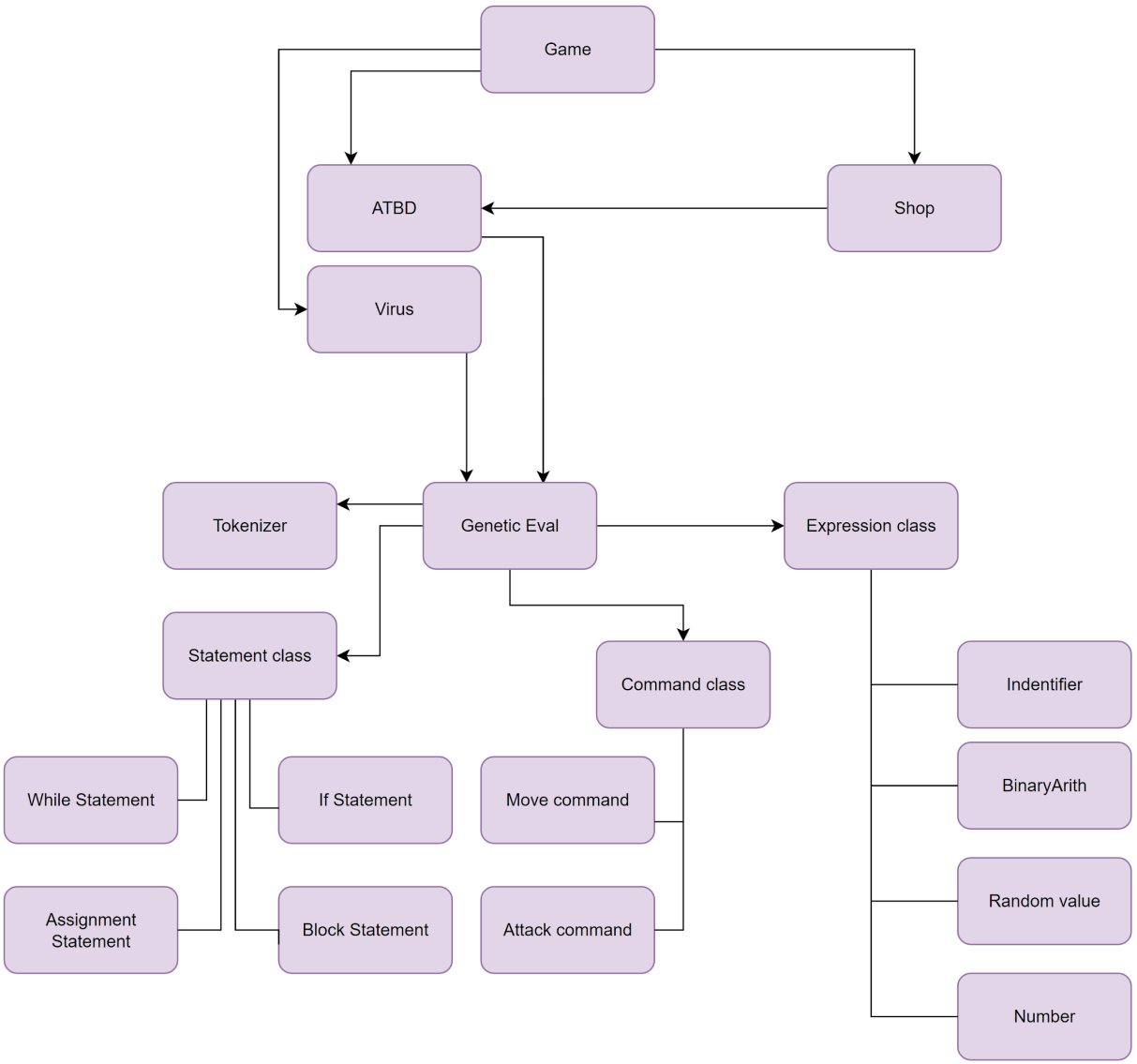
- implements from Evaluable
    - class **Number**
        - ri :The number must be a nonnegative integer.
        - number : int
        - eval() : int
            - return the value of the number.
    - class **Identifier**
        - ri : The identifier's name must not be a reserved word.
        - bindings : Map<String, int>
        - variable: String
        - eval() : int
            - return the value of the variable.
    - class **BinaryArith**
        - ri: Division or Modulo by zero is not allowed
        - 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**
        - ri: statements can have zero or more *Executable*
        - 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**
        - ri: 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
        - 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 to store in the unit
    - ri: genetic code must be appropriate grammar format
    - tkz : **Tokenizer**
    - unit : Unit
    - bindings : Map<String,int> - bindings of this unit.
    - instance : static GeneticEvaluator
    - 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

Dependency Graph:



uses →

includes —

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 - used in frontend such as if player wins a game so frontend must change state from "playing" to "win" state to display win scene

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
  - SortedSet to store empty tile(sort to spawn viruses in certain side of field)
  - Array[] to store set of same data

- 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.
  - Displaying is not real time because game data cannot be sent smoothly.
  - The more dimensions of the field in the game , the slower rendering.

Tools and Languages

- Tool
  - Display and Game Element Design : Figma
  - Backend : Java , Spring Framework, Spring RESTful API (VScode, IntelliJ)
  - Frontend : HTML, CSS, Javascript, TypeScript, React (VScode)
  - Communication : Discord

What we learn from designing and implementing

- redesigning action could still occur even 1 day before the deadline.
- the design and implementation was fine until the concurrent nation attack
- throwing exception made exiting method easy (and also debugging)
- time and tide wait for no man. We must manage time and schedule efficiently. If there is something you can’t do, take more time with those things.

-

Testing

- outline what you tested
    - Blackbox testing
        - play test
        - configuration test
        - frontend test
        - API test (including sending and receiving data from API)
    - Glassbox testing
        - test statement and branch coverage of each methods
- how did you test your various parts mentioned above?
    - play test
        - manual test for each corner case that can make program not working properly
        - test player input and control provided by frontend
        - test all win and lose conditions
        - test game balance(very hard and time consuming)
    - configuration test
        - test negative integer input.
        - test minimum and maximum input for each line.
    - frontend test
        - When we create a new component , we test it on the website by simulating and using them to see if they work normally or not.
        - try to interact with the website in different behavior and abnormal ways to see whether the website can handle that interaction.
    - API test (including sending and receiving data from API)
        - At first, we will test by sending some simple data into the API and let the frontend and backend fetch it.
        - After that, we will test how fast and efficiently the frontend and backend send and receive data from API
        - Then, we will simulate the data flow between frontend and backend and see how it works.
    - statement and branch coverage of methods will explain with example
        - for example MOVE method
            - we test move in all direction
            - move out of field[][] range
            - move to occupied tile
            - move penalty
            - move from genetic code
        - senseNearby and senseClosest
            - sense in all directions

- correctness of sense (close, far, multi unit around caller)
- geneticEvaluator calling this method (can they work together correctly)

What did your group learn from testing during the entire project?

- *“Program will not have any bug if you don’t test it” -good Ochin’s pupil*
- Program testing can be used to show the presence of bugs, but never to show their absence! - Edsger W. Dijkstra
- There exist bugs after most testing
- redesigning sometime happens after testing thus leads to better design and implementation

Work Plan

- explain the order you developed the various project components

  there are no order of developing each component was developed parallelly

  - game class
  - parser and evaluator class
  - API
  - frontend website
- explain the role of member(s) in your group responsible for the project
  - Pawaret Dilokwuttisit - Evaluator class, API and frontend website, advisor (help members in other parts).
  - Ratchanon Niwat , Saranphot Chesrichai- the whole game(all classes that are not Parser and Evaluator)
    sometime we ทำโค้ดด้วยกัน by streaming in discord ช่วยกันคิดโค้ด some codes(method ย่อยๆ) are written separately.
  - Ratchanon Niwat - testing and game balancing
  - Saranphot Chesrichai - redesign some method and field,debug bugs in various classes.
  - did your group change the division of work during the project? if so, why, what was the change, and how did you end up with that change? did the change work out as expected?
    - 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.
- identify any key dependencies between different implementation steps, where one team member could start an implementation task until another team member had completed their assigned task
  - parser & evaluator -> Game
  - Game <--> API (actually they are not really dependent but the communication between them are)
- what did your group learn from the process of dividing up the work, including changes along the way?
  - ใช้งานให้ถูกกับคน ใช้คนให้ถูกกับงาน ใช้เวลาและทรัพยากรให้เหมาะสมกับงานและคน - ปวเรศ ดิลกฯ
  - Good communication is the key to success

Known Problems

- detail any known problems with your specification, design, or implementation
    - Playing games on the web using Spring Framework in project specification is too specific. Maybe we can use other things like game engines etc.
    - Implementation of class maybe complex and confusing (not according to design plan)
- did you finish the entire project on time? if not, why not? and give an estimate as to when you think you will have the entire project completed, and why. (you don't need to keep doing the project after this point, but it would be nice to estimate how far you think you are from completion)
    - We can finish the entire project on time. The game can be playable now.

Comments

    - Learning API is hard and tough. But if we can pass that point, we can finally find happiness.
    - Assigning an assignment near deadline made me byebye
    - game balancing is very hard when virus can spawn if it kill atbd not to mention lifestealing and kill gaining and all these into 3 different viruses and antibodies
- you might address these questions:
    - how much time did you spend on the design?
        - 4days
    - how much time did you spend on the implementation?
        - from assigned to presentation
    - how much time did you spend on testing?
        - about 30% of the whole time spent
    - what advice should we have given you before you started?
        - API and how to created it, Spring Framework Tutorial
    - what was surprising about the project?
        - game balancing(not only this project) is surprisingly hard.
    - what was hard about the project?
        - game
        - frontend and API (i have to dig my skill that i buried first semester up to this project)
    - what did you like or dislike about the project?
        - At first i think i like the project , but now i like the ice cream

        - jump attack required ahead
        - didn’t expect strong foe
        - i dislike the slow announcement of project
    - what would you change about the project?
        - nothing exactly (gonna watching to Ms Marvel)