Progress Report

Architecture

* Description format has been changed a little bit. sorry

* Since so many methods have been added and many methods' detail have been changed after the design so we decided to write new report

Interface

- 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);

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;

        - Unit’s health
    - *int* maxHp;
        - Unit’s maximum health(hp with not exceed maxhp)
    - *int* Atk;
        - Unit’s potential to inflict damage
    - *int* lifeSteal;
        - for Virus it is health restoration when attacking
        - for ATBD it is Kill gain
    - *int* cost;
        - storing cost of Unit
        - currently not used in Virus but who knows?
    - *int* moveCost;
        - move penalty of Unit
    - *int* attackRange;
        - maximum shoot range
        - use in shoot()
    - 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
- 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
        - 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

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***
    - 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

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.
    - currency : int
    - Map<ATBD,Boolean> true if currency >= ATBD.cost this field will affect displaying(frontend) -Changed because it is not necessary to store ATBD and boolean we can use int instead
    - Map<Integer,Boolean> status
        - this field will communicate to API
        - Integer is for the cost of all ATBDs.
        - Boolean will be true if currency >= cost.
    - updateBool():for loop of the Map to update the boolean value in each time unit This method will be called by ***Game*** class
    - has been renamed to updateStatus in real implementation
    - Shop(int[] cost)
        - Constructor (singleton ly)
        - 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 the config file is read
        - will be called in ***Game***.Initialize()
- class ***Game*** : **Singleton**
    - **Fields :**
        - ***static int*** m,n;
            - the vertical and horizontal length of the field.
        - ***static*** Unit[][] field;
            - used to store units in each position.
        - ***protected static final*** String inFile;
            - input file that contains config.
        - ***static int*** initialATBDCredits,atbdPlacementCost,initVirusHP,initATBDHP , initVirusATK , initVirusLifeSteal,

initATBDATK, initATBDLifeSteal , atbdMoveCost , atbdCreditsDrop;
    - to store value from the config file.
- ***static double*** spawnCount;
    - count the time that a new virus will spawn.
- ***static double*** virusSpawnRate;
    - the number of viruses that spawn in each time unit.
- ***static*** SortedSet<String> emptySlot;
    - the set that contains the empty slot in the field.
- ***static*** GeneticEvaluator g;
    - create the genetic evaluator.
- ***static*** List<Unit> order;
    - contains the unit in the field.
- ***static*** List<Unit> virusOrder;
    - contain the viruses in the field.
- ***static*** List<Unit> atbdOrder;
    - contain the antibodies in the field.
- ***static*** Objective Objective;
    - count the viruses that the player has to kill.
    - to tell state of game
- ***static int*** virusLimit;
    - the limit of the viruses in the field
- ***static*** Shop shop;
    - create the shop.
- ***static*** Unit gangster;
    - virus type 1.
- ***static*** Unit pistolDude;
    - virus type 2.
- ***static*** Unit sniper;
    - virus type 3.
- ***static*** Unit[] viruses;
    - array of viruses.
- ***static*** Unit Merci;
    - antibody type 1.
- ***static*** Unit Ana;
    - antibody type 2.
- ***static*** Unit Lucio;
    - antibody type 3.
- ***static*** Unit[] Atbds = {Merci,Ana,Lucio};
    - array of antibodies.
- ***static int*** virustemplate;

        - size of virus array, used to initialize the game.
    - ***static int*** atbdtemplate;
        - size of antibody array, used to initialize the game.
    - ***static int***[] cost = {Merci.getCost(),Ana.getCost(),Lucio.getCost()};
        - get the cost of each antibody.
        - to contruct Shop
    -
- Method :
    - ***public static void*** initObjective(***int*** maxElim);
        - set initial objective of the game
    - ***public static void*** notifyReachElim();
        - notify the player if the player wins or loses.
    - ***public static*** Pair<Integer,Integer> randomTile();
        - return a random empty tile in the field.
    - ***public static void*** add(Unit unit, Pair<Integer,Integer> position)
        - root method of adding unit
        - add units into the list of units and field[][].
    - ***public static 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 static 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 static void*** remove(Pair<Integer,Integer> position)
        - root method of removing unit
        - remove the unit that stores in that position of field[][].
    - ***public static 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 must pay the health for this method.

- ***public static void*** move (Unit u, Pair<Integer,Integer> destination)
    - root method of moving unit
    - move the unit to the destination without cost.
- ***public static 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 static 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 static void*** visualize()
    - visualize the game on the command line.
- ***public static int*** senseClosestVirus(Unit u)
    - find the closest virus from the input unit.
- ***public static int*** senseClosestATBD(Unit u)
    - find the closest antibody from the input unit.
- ***public static int*** senseNearby(Unit u, String direction)
    - find the closest unit from the input direction of the input unit.
- ***public static 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 static void*** updateShop()
    - update status of the shop.
    - call shop.updatestatus()
- ***static*** Unit createNewVirus(***int*** n)
    - create a new virus type n (there are 3 different types of virus).
- ***static*** Unit createNewATBD(***int*** n)
    - create a new antibody type n (there are 3 different types of antibodies).
- ***public static void*** Initialize()
    - initialize the value of fields.

- ***public static void*** Update()
    - update the game while the game is not over.

Parser Class:

- implements from Evaluable
    - class **Number**
    - class **Identifier**
    - class **BinaryArith**
    - class **SensorExpression**
    - class **RandomValue**
- implements from Executable
    - class **BlockStatement**
    - class **IfStatement**
    - class **WhileStatement**
    - class **AssignmentStatement**
    - class **MoveCommand**
    - class **AttackCommand**
- implements from token
    - class **Tokenizer**
- class **GeneticEvaluator**

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 various parts of the project in addition to what you described in the last report?
  -tested by static void main calling visualize method to see what happens in the field[][].
    - We tested every method after we wrote them in terms of branch coverage for example
    - most recent testing is game loop and it has been successful
        - every time unit, the game spawn virus from random empty tile, unit can execute their program from genetic code,unit destructed properly and game loop end after the objective reached.
- describe test cases you have used or will be used for testing
  We tested every method after we wrote them in terms of branch coverage for example

    - all possible move direction and the corner such as move unit out of the field[][]
    - add,remove, move logic normal case(in field) and corner case (not in field, position occupied by another unit) tested.

        - randomTile and emptyTile tested in terms of correctness.
        - shoot logic(hp mod?, shoot the opposite organism?, shoot correct organism, destruct properly after dead)
        - create the unit until the field has no empty tile.
        - the most recent game loop testing
            - init objective 100(kill 100 viruses to win the game)
            - creating a very strong ATBD that instantly kill any viruses in its attack range using example genetic code
            - randomly spawn virus every time unit
            - observe by visualize method
            - the ATBD kill viruses
            - the game loop run until the objective is reached
            - *This test proves correctness of genetic code parsing, shoot method, destructing unit, move method , objective, shop currency updating, randoming empty tile. bugs found : randomtile position extracting, concerrentmod Exception occur in unit program executing loop

- What did your group learn from testing that you have not talked about in the last report?
    - constantly testing newly written code reduces the overall bug when testing the whole program.
    - exceptions throwing is the main protagonist of writing codes

Work plan

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

- Pawaret Dilokwuttisit - Advisor
- Ratchanon Niwat , Saranphot Chesrichai- the whole game details in Obranch of the repo
  sometime we ทำโค้ดด้วยกัน by streaming in discord ช่วยกันคิดโค้ด
  some codes(method ย่อยๆ) are written separately.

Did your group change the division of work? If so, why, what was the change, and how did you end up with that change?

- Nothing changed this time

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

- “Put the right man in the right job”
- Good management leads us to the goal.

Pawaret “DarkTXYZ” Dilokwuttisit , 24 Feb 2022   -