CPE 200 Semester 2 2021
Project CAREN
Design Overview Document

Baangkok Vanijyananda, Ukrit Kosonsomboon, Nonthawat Kongsichai

Version of March 16, 2022 (Finished implementation)

1 Changes

Change Log	
Date	Changes
Feb 8, 2022	> Initial version
Feb 16, 2022	> Added classes: ShopController, GameController, AreaController, WaveController, ZoomController to 3.1-Model, 3.2-Class Hierarchy > Moved class: Button to front-end. > Rewrote 4.2-Module Dependency Diagram and 4.3-UML Diagram to match Class Hierarchy > Added 4.4-What we learned from designing parser > Added Testing Report to the bottom of 6-Testing > Rewrote 2nd paragraph of 7.3-Key Dependency > Added more problems and resolved problems to 8-Known Problems > Added more what did we liked... to 10-Comments
Feb 24, 2022	> Added class: InventoryController to 3.1-Model, 3.2-Class Hierarchy > Moved class: ZoomManager to front-end > Removed class: ZoomController, Wave > Rewrote 4.2-Module Dependency Diagram and 4.3-UML Diagram to match Class Hierarchy > Changed 4.4-What we learned from designing parser to 4.4-What we learned from designing and implementing > Added more to 6.2-Testing Report > Added a paragraph about controllers 7.3-Key Dependency > Added more problems and resolved problems to 8-Known Problems > Added to Time spent, What we liked about the project in 10-Comments

Change Log	
Date	Changes
March 16, 2022	> Added classes: Vector2, ButtonObject, WaveInfo to front-end to 4.2-Class Hierachy > Removed all controller classes other than GameController from front-end from 4.2-Class Hierachy > Added classes: GameHandler, Config, variations of ApiData to back-end to 4.2-Class Hierachy > Made GameHandler the only singleton in back-end in 4.2-Class Hierachy > Removed some unused Node classes, GameObject, Button from back-end from 4.2-Class Hierachy > Added class: Config to 5.1-Key Classes and Interfaces > Merged subsection 6.3 into 6.2 > Added more to 6.2-Testing Report and Coverage Analyzation > Added more to 7.3-Key Dependency > Added new subsection: 7.4-What did we learn from the process of dividing up the work > Added more to 8-Known Problems > Added more to 10-Comments > Added more to 11-Learning Resources

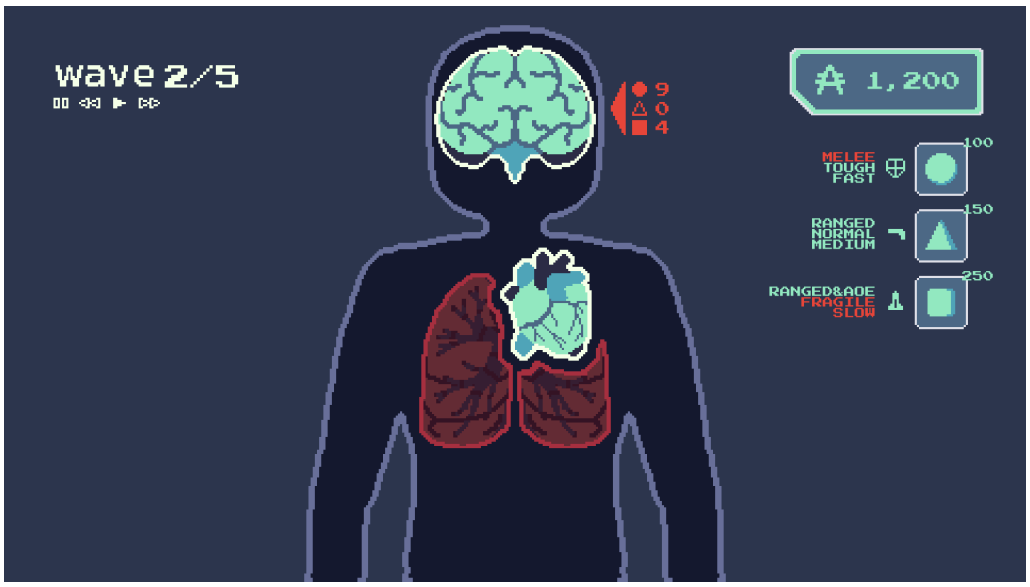
2 Introduction

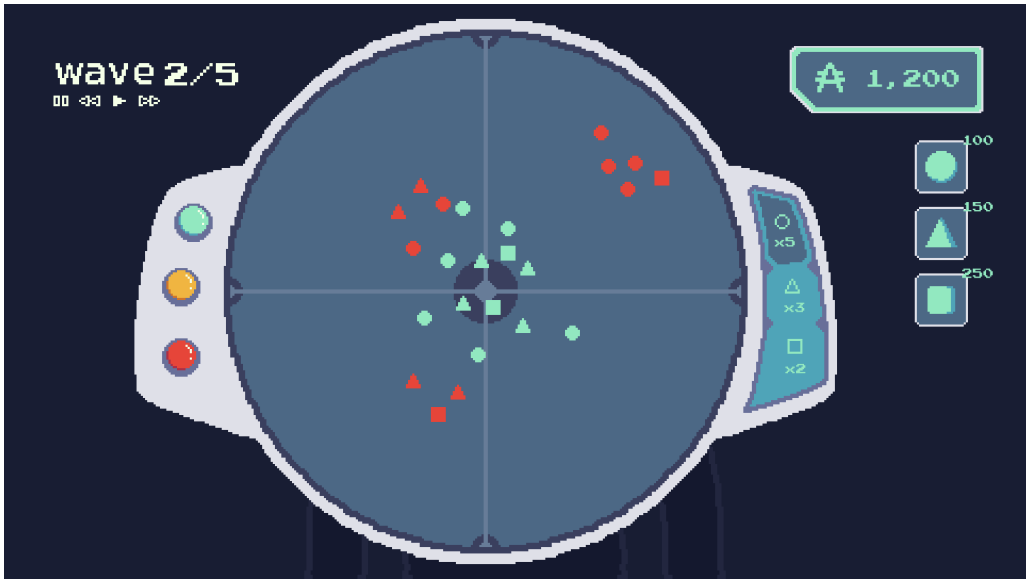
CAREN is a simulation game of a battle between viruses and antibodies in your body. Both have their unique genetic codes which determine their behavior. Genetic codes are written using a specified grammar as per stated in the project specification. The game must be run on a web page with a Java back-end which handles all the non-visual computation of the game.

3 Game Overview

3.1 Figma Prototype

Figma Prototype Draft: CPE200-Project





3.2 Gameplay Overview

Our version of CAREN have 3 vulnerable areas you must defend: brain, heart, and lungs. Each area starts with some antibodies. In each wave, various kinds of viruses will attack random areas. The player can choose to spend antibody credits to place new antibodies for a full cost, or move existing ones for a reduced cost. There are 3 types of antibodies and viruses: melee, ranged, and aoe. The player must not let the number of antibody in the area reach zero other wise the area is deemed lost. If all areas are lost, the players loses. If all area survives through all the waves, the player survives the pandemic.

4 System Architecture

4.1 Model

- Speed Multiplier
- Current wave
  - Count
  - Target Area
  - Total virus count for this Wave
  - Virus type for this wave
- Antibody credit
- Store
  - Units
    - Name
    - Description
    - Cost
    - Stats
    - Genetic Code
  - Inventory
    - Units
    - Unit Count for each unit

- Area
  - Alert System
    - Virus per Antibody Alert Threshold
  - Area Overrun State
  - Antibody Count
  - Virus Count
  - Antibody and Virus
    - Health
    - Manual Move Cost
    - Attack Damage
    - Life Steal
    - Move Speed
    - Type (Melee/Ranged/AOE)
    - Attack Range (for Ranged)
    - Size
    - AOE Size (for AOE)
    - Genetic Code
    - Position
    - Which virus to spawn when dead (Antibody)
    - Credit Reward (Virus)

4.2 Class Hierarchy

Interface are denoted with: >

Class are denoted with: -

For detailed class structure, please see *4.3 UML Diagram*.

4.2.1 Front-End

- TextObject
- ImageObject
- GameController
- Vector2
- ButtonObject
- WaveInfo

4.2.2 Back-End

- GameHandler (Made singleton with spring)
- Game
- GameController
- Config
- Shop
- ShopController
- Area
- AreaController
- Inventory

- InventoryController
- WaveManager
- WaveController
- TimeManager
- TimeController
- Tokenizer
- BehaviorEvaluator (Parser)
- GeneticCodeManager

> Node

  - ProgramNode
  - MoveCommandNode
  - AttackCommandNode
  - AssignmentStatementNode
  - IfStatementNode
  - WhileStatementNode
  - NumberNode
  - VariableNode
  - RandomNumberNode
  - BinaryArithmeticNode
  - SensorNode

> variations of ApiData

- UnitFactory (Factory)
- Unit
  - Antibody
  - Virus

4.3 Tools

4.3.1 Languages and Libraries

React.js: Constant update up-to 60 fps, JavaScript compatibility.

Typescript: Strict syntatical static typing superset of JavaScript.

Node.js: JavaScript runtime and mainly for drawing to <canvas> tag.

Maven: Java dependency manager.

HTML5: <canvas> tag for drawing shapes and images similarly to a game engine's window.

CSS: Beautify and customize the web page.

Spring: Framework to connect the front-end to back-end via api.

Java: Object-Oriented back-end.

4.3.2 Programs and Services

Visual Studio Code: HTML, CSS, JavaScript , Typescript IDE.
IntelliJ IDEA : Java IDE.
Aseprite: 2D Pixelart Program.
Figma: Front-end Prototyping Tool.
Trello: Project Schedule Management Tool.

4.4 What we learned from designing and implementing

- Front-end
  - Making a prototype project would help a lot when making a project with new tools/workflow. This caused front-end to have long codes (but not spaghetti though!)
  - I should thoroughly study <canvas> tag beforehand so i don't have to implement onclick in each object by myself.
  - Using spring, there really isn't a way for back-end to notify() front-end, and front-end can only poll from back-end, not the other way round.
- Parser
  - A well design Node system will help better when making changes.
  - It's good to separate concerns to different classes, but not too much.
  - Some things are hard to test with JUnit. Or to say other testing method(visually seeing) is easier.
  - If(DEBUG) print() is nice.
  - Parser is complicated, if not managed well it will become hard to use.
  - Using floating-point instead if integer in the AST sometimes produced unintended result such as: dividing '/' produce decimal points which causes wrong boolean evaluation. eg. if(1/2) have a true value instead of intended false because 1/2 is 0.5 which is still > 0. We fixed that by literally removing the decimal points produced by dividing.
- Game
  - เขียนการวนลูปของ state ต่างๆให้ดีและครอบคุม เพราะในทุกๆ loop มันจะมีบ้าง method ที่จะต้องมีในทุกๆที่ ไม่งั้นมันจะทำให้แต่ละ state แทนที่จะหยิบ Unit ได้บ้างไม่ได้บ้าง ต้องหยิบได้ในทุกๆ state
- WaveManager
  - สร้าง Wave ครั้งเดียวละก็แค่หยิบนำไปใช้เลยไม่ต้องสร้างทุกครั้งที่ต้องการจะใช้ มันง่ายต่อการใช้งานมักๆ เพราะเราปล่อยเป็น Wave ทำแบบนี้ได้สดวกดี
- Unit

  - จากการเขียน Unit ที่ผ่านมาทำให้ได้เรียนรู้หลายๆอย่าง ทั้งในเรื่องของการลดการ duplicate โค้ด ที่ไม่จำเป็นต้องเขียนหลายๆที่แล้วยุบเป็น method หรือเก็บไว้ใน variables ต่างๆได้ หรือไม่ว่าจะเป็นการเขียน Factory ขึ้นมาเพื่อให้มันสร้างได้ง่าย นำไปใช้งานสะดวก
- AreaController and other controllers
  - จากการเขียน Area และเหล่า Controller ของ Class ต่างๆ ก็ได้เรียนรู้การสื่อสารกับ Front-end เรียนรู้การใช้ Spring พวก GetMapping PostMapping อะไรพวกนี้มาค่อนข้างเยอะเพราะว่าไม่เคยรู้มาก่อนเลย และที่เอา Area ขึ้นก็เพราะว่า Area เป็น Class แรกที่ทดลองใช้สื่อสารกับ Front-end สำเร็จ ”AreaController” คือจุดเริ่มต้นของความสำเร็จเลยก็ว่าได้

5 Detailed System Design

5.1 Key Classes and Interfaces

GameHandler: The class for managing Game instances.

Game: The main class that contains the game loop and connects everything together.

Config: The class for reading and storing the configuration file.

Shop: The class for managing antibody credits and the purchasing of Antibody.

Inventory: The class for managing Antibody storing, placement, and re-positioning.

WaveManager: The class for managing Virus waves.

TimeManager: The class for managing speed multiplier of the time step.

ZoomManager: The class for managing zoom state.

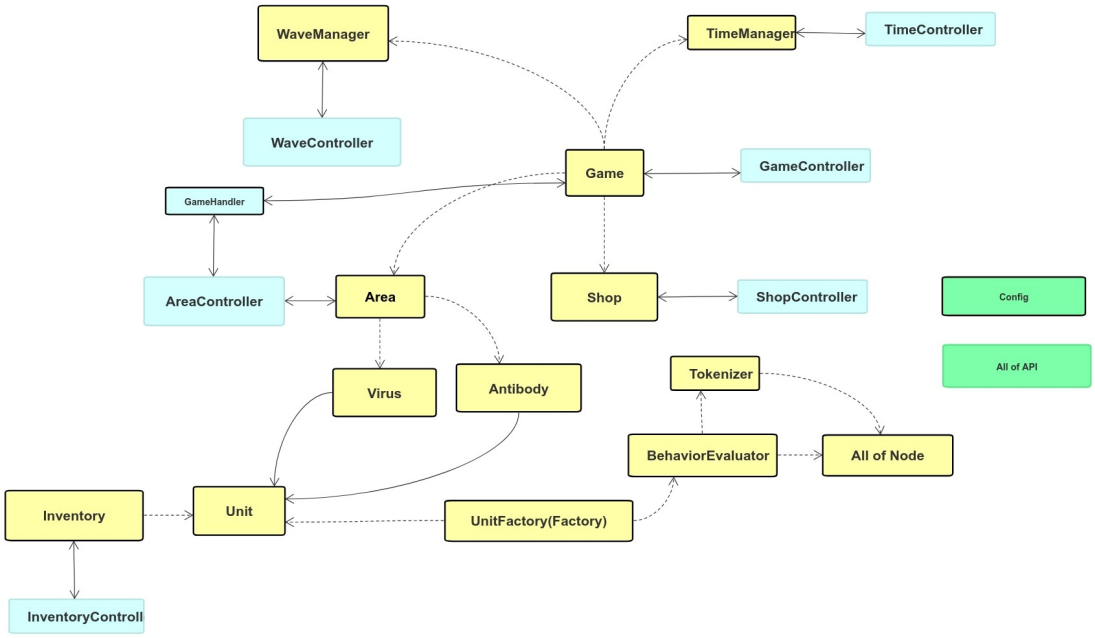
BehaviorEvaluator(Parser): The class that's used to create AbstractSyntaxTree to evaluate the genetic code written from the grammar.

GeneticCodeManager: The class for managing genetic code for each Unit type.

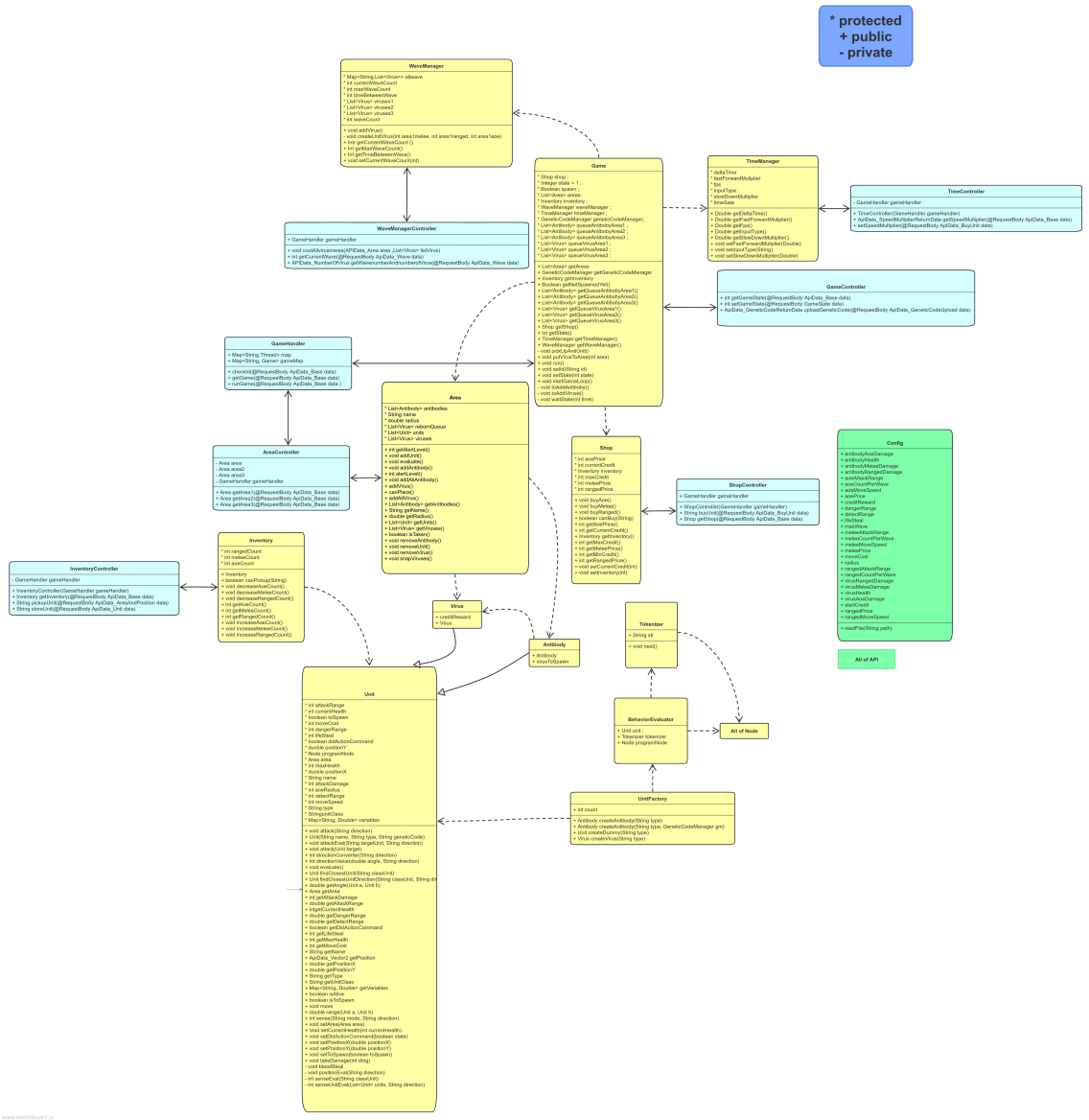
Node: The interface for all nodes that will be generated from the genetic code to create an AbstractSyntaxTree.

Unit: The parent class for Virus and Antibody.

5.2 Module Dependency Diagram



5.3 UML Diagram



5.4 Important Rep Invariant

Shop: Antibody credits must not exceed 9,999, go below 0.

TimeManager: Multipliers must not be a number below zero.

ZoomManager: Zoom level must remain between min zoom and max zoom level.

BehaviorEvaluator: Program node must not be null.

Unit: Current health must not exceed max health, Position must remain within the circle radius.

Inventory: Each type unit count must not go below 0.

5.5 Code Design

We design our code architecture with 3 things in mind:

1. **High Readability** - The code should be as readable as possible, names may be longer but that makes it easier to understand the code and improve upon this project.
2. **Separation of Concerns and Ease of Use** - The code should be separated in according to it's purposes, so that during testing we can pinpoint the cause of bugs more easily and the code can be reused else where where the similar functionality are needed.
3. **Efficiency** - The code should be made to be efficient to the point where the gameplay feels smooth. Highest efficiency is not needed for the current spec of the project but that can be an improvement for the future.

   There is currently a low-framerate problem, caused by multiple instances of Unit evaluating in one time step.

5.6 Design Patterns

Factory Method: There are many Unit types with different stats. The UnitFactory handles that. Same with multiple types of Node, NodeFactory handles that.

Singleton: Many of our classes can only have one instance.

Interning: We might have to do this since there could be more Game instance when more than one person is playing. This is in GameHandler.

State: Buttons display and act according to its state.

Iterator: Tokenizer is an iterator for genetic codes. Front-end fetching data need to draw from back-end is similar to iterator.

6 Testing

6.1 Testing plan

– Using JUnit to test methods

  – Game

    before and during coding : ก่อนที่จะรันเกมก็ต้องทดสอบก่อนว่าสิ่งที่ออกแบบมานั้นมันสามารถเอามารันจริงได้หรือเปล่า

    after and during coding : ทดสอบเกมโดยการลองรันดู ระบบทุกอย่างในเกมที่ได้ออกแบบมา ลองเล่นดูจริงๆ

  – Shop

    before and during coding : Shop ที่กำลังจะเขียนนั้นสามารถซื้อของเกินเงินที่มีอยู่ได้หรือมั้ย ซื้อของแล้วเงินลดมั้ย

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลองเข้าไปซื้อของ(กดซื้อโดยที่เงินไม่พอ ดูว่าเงินที่กดซื้อไปลดมั้ย)

- Area

  before and during coding : Area ที่กำลังจะเขียนนั้นสามารถนำ Unit(Antibody and Virus)ไปวางบน Area ได้จริงมั้ยและ Unit แต่ละตัวสามารถเดินออกนอก Area ได้มั้ย

  after and during coding : หลังจากเขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่ แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลองนำ Unit ไปวาง(ดูว่า Unit ปรากฏบน Area มั้ยแล้วเดินออกนอก Area ได้มั้ย)

- Inventory

  before and during coding : Inventory ที่กำลังจะเขียนนั้นสามารถเก็บของได้จริงหรือไม่ แล้วถ้าในนั้นไม่มีอะไรอยู่เลยมันจะสามารถดึง Unit ออกมาได้หรือไม่ และ หากดึง Unit ที่เก็บไว้หมดแล้วยังจะสามารถดึงซ้ำออกมาได้อีกหรือไม่

  after and during coding : หลังจากเขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลองเข้าไปดูที่ Inventory(ลากเข้าลากออก Unit)

- WaveManager

  before and during coding : WaveManager ที่กำลังจะเขียนนั้นสามารถจัดการกับ Wave ได้จริงหรือปล่าว ไป Wave ต่อไปแล้ว ตัวเลขของ Wave เปลี่ยนหรือไม่

  after and during coding : หลังจากเขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลองเพิ่มลด Wave เพื่อดูว่าตัวเลข Wave เปลี่ยนไปตามความเป็นจริงหรือไม่

- Wave

  before and during coding : Wave ที่กำลังจะเขียนนั้นสามารถจัดการกับ Virus ที่จะส่งออกมาแต่ละ Wave ตามที่กำหนดไว้ได้หรือไม่ จะส่งมาขาดหรือเกินหรือไม่

  after and during coding : หลังจากเขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลองเพิ่มลด Wave เพื่อดูว่าจำนวน Virus ที่ส่งออกมาตามที่กำหนดมั้ย แล้วขาดเกินหรือไม่

- TimeManager

  before and during coding : TimeManager ที่กำลังจะเขียนนั้นสามารถจัดการกับความเร็วในเกมได้จริงหรือไม่ หากสั่งให้ Fast Forward แล้วกระบวนการทั้งหมดในเกมจะเร็วขึ้นจริงมั้ย หากสั่งให้ Slow down แล้วกระบวนการทั้งหมดในเกมจะช้าลงจริงมั้ย และสุดท้ายหากสั่งให้ Pause เกมจะหยุดทุกอย่างได้จริงมั้ย

  after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลอสั่งการทั้ง Fast forward, Slow down, Pause แล้วดูผลลัพธ์ว่าเกิดอะไรขึ้นบ้างแล้วตรงตามคำสั่งที่ต้องการหรือไม่

- ZoomManager

before and during coding : ZoomManager ที่กำลังจะเขียนนั้นสามารถจัดการกับการ Zoom ในเกมได้จริงหรือไม่ ทั้ง Zoom in(สามารถ Zoom เข้าไปเฉพาะจุดที่ต้องการได้หรือไม่) และ Zoom out(สามารถ Zoom ออกจากจุดที่ Zoom อยู่ ณ ตอนนี้ได้หรือไม่) ขณะที่ Zoom อยู่พื้นที่รอบๆที่ไม่ได้ Zoom ยังทำงานปกติหรือไม่

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วมีบัคใหม่ๆขึ้นมาบ้างรึปล่าว โดยการลอง Zoom in , Zoom out ดู แล้วดูผลลัพธ์ว่าสามารถทำได้จริงมั้ย แล้วพื้นที่โดยรอบยังทำงานปกติรึปล่าว

– Tokenizer

before and during coding : Tokenizer ที่กำลังจะเขียนนั้นสามารถแปลง Genetic Code ที่ให้ไว้กลายเป็น Token ได้หรือไม่

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าว โดยการลองป้อน Genetic Code ไป แล้วก็ลอง Print ออกมาดูว่า มันสามารถอ่านได้ถูกต้องและครบถ้วนหรือไม่

– BehaviorEvaluator(Parser)

before and during coding : BehaviorEvaluator ที่กำลังจะเขียนขึ้นมานั้นสามารถ Parse Grammar ที่มีได้หรือไม่

after and during coding : หลังจากที่เขียนเสร็จแล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าวแล้วลอง Parse Grammar ที่มีอยู่ทั้งหมด แล้วดูผลลัพธ์ว่าใช้งานได้จริงหรือไม่

– Node

before and during coding : Node ที่กำลังจะเขียนนั้นสามารถ ใช้เป็น Node ใน Abstract Syntax Trees ได้จริงหรือไม่

after and during coding : หลังจากที่เขียนเสร็จแล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าวแล้วลอง ใช้ Node ต่างๆใน Abstract Syntax Trees ดู แล้วดูผลลัพธ์ว่าใช้งานได้จริงหรือไม่

– UnitFactory(Factory)

before and during coding : UnitFactory ที่กำลังจะเขียนนั้นสามารถใช้สร้าง Unit(Antibody and Virus) ได้จริงหรือไม่ แล้วสามารถใช้สร้างอย่างอื่นได้อีกหรือไม่(ไม่ควรได้)

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าว โดยการลองสร้าง Unit ดูหลายๆ Unit ทั้ง Antibody และ Virus แล้วลองใช้สร้างอย่างอื่นด้วย(ไม่ควรได้)

– Unit

before and during coding : Unit ที่กำลังจะเขียนขึ้นมานั้นสามารถสร้างเป็น Unit ได้จริงหรือไม่ สามารถเดินได้, โจมตีได้, ตายได้ หรือไม่

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าว โดยการลองสร้าง Unit ขึ้นมาหลายๆตัว และป้อนคำสั่งต่างๆ เช่น บังคับให้มันเดิน ให้มันตีกัน เป็นต้น

  – Antibody

before and during coding : Antibody ที่กำลังจะเขียนขึ้นมานั้นสามารถสร้างเป็น Antibody ได้จริงหรือไม่ สิ่งที่ Antibody ต่างจาก Unit ธรรมดาก็คือเวลาตายแล้วจะกลายพันธุ์เป็น Virus ก็ดูว่าเวลาตายแล้วจะกลายเป็น Virus ได้มั้ย

after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าว โดยการลองสร้าง Antibody ขึ้นมาหลายๆตัวแล้วลองให้มันตีกันเองก่อน แล้วก็ดูผลลัพธ์ว่าตายแล้วกลายพันธุ์เป็น Virus ได้มั้ย

- Virus

  before and during coding : Virus ที่กำลังจะเขียนขึ้นมานั้นสามารถสร้างเป็น Virus ได้จริงหรือไม่ เวลา Virus ตายแล้วจะได้เงิน(Antibody Credit) มั้ย

  after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือไม่แล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าว โดยการลองสร้าง Virus ขึ้นมาหลายๆตัวแล้วให้มันตีกันเองก่อน แล้วก็ดูผลลัพธ์ว่าตายแล้ว Antibody Credit เพิ่มขึ้นมั้ย

- Compatibility Testing between Front-End and Back-End

  before creating project : ต้องทดสอบก่อนว่า Front-End กับ Back-End ที่เราเลือกนั้นสามารถเข้ากันได้หรือไม่ โดยการลองสร้างโปรเจคจำลองขึ้นมาแล้วก็ทดสอบดูเลย(หาโปรเจคจำลองได้จาก Template ใน Internet)

  during and after finishing project : หลังจากที่เราทดสอบในช่วงแรกแล้ว ก็ต้องลองกับของจริงโดยการนำโปรเจคของเรานี่แหละไปทดสอบว่ามันเข้ากันได้หรือไม่(ควรจะได้) แล้วก็ดูผลลัพธ์ว่ามันเข้ากันได้จริงหรือไม่

- Deploy web

  before deploy : ลองทดสอบดูว่า Web-host ที่เราเลือกมานั้นสามารถ run project ของเราหรือไม่

  after deployed : หลังจากที่ได้ Deploy ลงบน Web-host แล้วก็ต้องดูว่ามีอะไรผิดเพี้ยนจากความต้องการของเราหรือไม่ โดยการเข้าไปดูใน Web ที่ Deploy ไปแล้วลองดูเรื่องรายละเอียดต่างๆ และลองเล่น Game ที่ได้ Deploy ลงไปดู

6.2 Testing Report and Coverage Analyzation

This part is the testing report. Detected bugs and issues will be in 8-Known Problems

- BehaviorEvaluator(Parser), Tokenizer, Nodes
  - We tested using a print method in Nodes which prints out the AST in a way that looks like a written code. If the AST logic seems legit, then it should be correct.
  - We tested using sample genetic code from the instructor and our own genetic codes.
  - On the expressions part of the AST, we tested with many different testcases, ranging from simple arithmetic to complex expressions. We tested both the correctly written expressions and the incorrectly written expressions.
  - We did not have many testcases for other parts since we decided that seeing the game run visually would help better at detecting bugs.

    We finished testing on march 16 and so far everything behaves correctly.

- Unit
  - สร้าง main มาลองทดสอบสร้าง Unit โดยใช้ UnitFactory เป็นตัวสร้างแล้วก็ลองใช้ method ต่างๆ แล้วก็ให้มันแสดงผลออกมาโดยการ print ออกมาว่าได้ผลหรือไม่
  - จากการ test ที่ทำไปข้างต้นทำให้รู้ว่า การใช้ Factory นี่มันดีจริงๆ
  - ใช้ JUnit ในการ Test method หลักๆ ก็ทำให้เจอ bug ต่างๆที่เราคิดไม่ถึงจาก TestCase ที่เราใช้ Test ใน JUnit
  - เจอว่า Unit บางตัวสามารถเดินทะลุ Area ที่ตั้งไว้(แก้แล้ว)
  - เจอว่า Unit ที่ตายไปแล้วยังโดนโจมตีอยู่ซ้ำๆ(แก้แล้ว)
  - เจอว่า Unit move อย่างเดียวแล้วก็ไม่โจมตี(แก้แล้ว)
  - เจอว่า Unit โจมตีไม่ได้ตามระยะของ type ตัวเอง ทุกตัวเข้าไปโจมตีระยะใกล้หมด(แก้แล้ว)

    Unit เสร็จแล้ว
- Area
  - ใช้ JUnit ในการ Test method หลักๆ ทำให้เจอ bug ที่เราคิดไม่ถึงจาก Testcase ที่ใช้ Test ใน JUnit
  - เจอว่า Unit ตายแล้วมันยังอยู่ใน list ของ unit ใน area นั้นๆ(แก้แล้ว)

    Area เสร็จแล้ว
- Shop
  - ใช้ JUnit ในการ Test method หลักๆ ทำให้เจอ bug ที่เราคิดไม่ถึงจาก Testcase ที่ใช้ Test ใน
  - จากการใช้ JUnit ไม่เจอ bug อะไร

    Shop เสร็จแล้ว
- Inventory
  - ใช้ JUnit ในการ Test method หลักๆ ทำให้เจอ bug ที่เราคิดไม่ถึงจาก Testcase ที่ใช้ Test ใน
  - เจอว่า Inventory เวลาที่หยิบ Unit ออกมาจาก Area จำนวนใน Inventory เพิ่มจริงแต่ว่าตัว Unit ตัวนั้นมันยังอยู่ที่เดิมทำให้กดหยิบได้เรื่อยๆ(แก้แล้ว)
  - เจอว่าเวลาหยิบ Unit ออกมาแล้ว เงินติดลบ(แก้แล้ว)
  - เจอว่าเวลาหยิบ Unit ออกจาก Inventory ลงไปใน Area แล้วมันไม่ยอมลงไป(แก้แล้ว)

    Inventory เสร็จแล้ว
- Game
  - ได้ลองเอา Virus ใน WaveManager มาลอง loop ว่ามาหรือเปล่าใน Main และแสดงการตีหรือการ test ของการ sene ว่าเจอกันหรือป่าว
  - ทดสอบ state ว่าถูกหรือป่าวการเข้าไปใน loop ของแต่ละ state ควรทำในสิ่งที่ต้องทำได้หรือป่าว โดยการเริ่มเกมส์ละ test
- Front-end
  - There are front-end controllers which are suppose to fetch data from back-end, some of APIs are not implemented yet so i was testing these controllers using dummy data. -> They're working fine with dummy data.
  - Now that APIs are finished, we can test the real frontend-backend interaction.

- I tested clicking-related methods using if(DEBUG) console.log() which is working just fine. -> If react runs at lower than 20 fps, the clicking will be less responsive. But we run it at 30 fps so it is fine.
- We will run it at 10 to 30 fps, depending on the 'play' 'fastforward' and numbers of Unit instances.
- Other tests are just testing by seeing if anything looks wrong, out of position. -> Nothing looks out of place, so far.
- Everything is working fine, so far.

7 Workplan

7.1 Work Allocation

Further details will be added in the future.

1 **All members**:
- Designing and All versions of Design Overview Document

2 **Ukrit**:
- Unit
- UnitFactory
- Area
- AreaController
- Shop
- ShopController
- Inventory
- InventoryController
- Config

3 **Baangkok**:
- Parser and Parser related classes
- GeneticCodeManager
- Front-end React and Typescript

4 **Nonthawat**:
- GameHandler
- Game
- GameController
- WaveManager
- WaveController
- TimeManager

– TimeController

**** Though there are designated roles. All of our members help each other whenever a significance problem occurs.**

7.2 Work Schedule

We use Trello to manage our work schedule. We recommend you see Trello instead of this document.

1 **Designing and Design Overview Document**: by Feb 8
2 **Front-end React, JavaScript**: by March 15
3 **Back-end Parser, Parser related class**: by Feb 12
4 **Back-end Game, Unit**: by March 16
5 **Back-end Other classes**: by March 16

Further details please see Trello.

7.3 Key Dependency

Key Dependencies are Parser and Parser related class, Game, and Unit because these classes are the back-bone of the project. These classes are the minimum requirement to run the game with working genetic code.

Controller classes are also important because it allows our back-end and front-end to communicate. Which helps with testing by visually seeing if anything goes wrong.

Front-end is a harder than anticipated because we decided to use <canvas> tag which require us to implement onClick in each element by ourselves.

1st Bottle neck happened in front-end development due to lack of knowledge about <canvas> development. This is not a problem.

2nd Bottle neck happened in back-end development due to one of our back-end developer having an earlier final exam than others. This is not a problem.

3nd Bottle neck happened in back-end development due to Game class finished slower than expected. But this is not a problem since the other 2 devs still have some testings they can do while waiting.

7.4 What did we learn from the process of dividing up the work

– It is crucial that in this process, everyone gets an equal amount of workload.
– It is crucial that the work plan is carefully arranged so that the least amount of bottleneck happens.

8 Known Problems

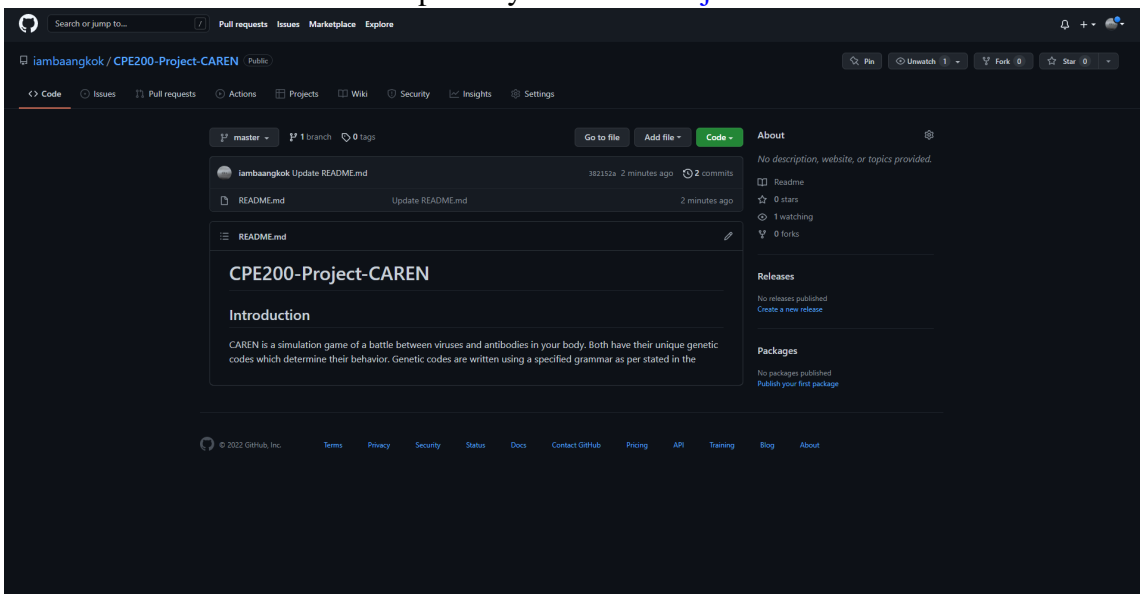
This part contains bugs, issues, and concerns we have.

- We are unsure if we could achieve the final project in time due to the increasing amount of concerns such as assignments and CPE208 Numerical midterm exam which takes place on February 13, 2022.
- We are unsure which web-hosting service we should use. We are aiming for itch.io since it supports web game but we're still not sure if our project will be able to run there. Other alternatives as we have researched so far are Heroku and Surge. We will not deploy on a web hosting service since the delay between fetching would be too much.
- We are aiming for a high framerate (30) application but we are unsure whether Spring can handle that much data request from the front-end or not. We ended up going with 10-30 fps.
- (fixed) CORS Problem มันคือปัญหาที่เรา fetch ข้าม server กันไปมาโดยปัญหาที่กลุ่มเราเจอก็คือตัวหน้าเว็บใช้เป็น server ของ localhot:3000 แต่ว่าค่าของ api ต่างๆที่ fetch มา (Spring) มันใช้ server ของ localhost:8080 จึงทำให้ Get, Post หรืออะไรก็ตามไม่ได้เลย แต่ปัญหานี้ก็แก้ไขเรียบร้อยแล้วครับ by using @CrossOrigin
- (fixed) @Component and @Autowired We are not sure how spring works behind the framework. @Autowired is one thing. But like how does everything works when we haven't create an instance of anything?? We'll study more into this.
- (fixed) Y axis positive direction is swapped in frontend and backend.
- (fixed) Unit refuses to attack, it only moves.
- (fixed) Unit only attacks when in close range.
- (fixed) Crash in GameHandler because of starting an already Alive() thread.
- (fixed) Unit going out of circle radius.
- (fixed) StoreUnit() when paused exploit.
- (fixed) Dead Unit still gets attacked.

There are other bugs but these are the most noticeable and time-consuming bugs.

9 GitHub Repository

GitHub Repository: CPE200 Project CAREN



10 Comments

Time spent: Around 40 on the design 40+20+40 on implementation and testing (both is done in parallel -> when we finished a portion enough for testing, we test it.) = total of 140 hours per person accounting for times we are together and times we are thinking by ourselves. This time also includes the time taken to type this document.

What we wish the instructor told us: The project specification should've been finished since the assignment of the project. The project could've been assigned earlier so we have more time to work on it without it getting close the the Final Exam days. But we understand the complications and that thinking up a project specification is not easy.

What was surprising about the design: With the separation of Model-View-Controller, everything suddenly feels more organized. The code will be shorter and less spaghetti-ish.

What was hard about the design: Deciding on how should we structure the Nodes. Deciding on how we should separate the Unit's by type (Antibody/Virus) and by class(Melee/Ranged/AOE). Testing how the the workflow between different tools will be and whether the workflow we wish to achieve will work or not. Learning ways to professionally do things, such as writing UML Diagram, writing a project document in LaTex, etc.

What did we like or dislike or wanted to change about the project:

bk: personally, i love this game we are making but i wished we didn't need to use Java as back-end for a small web-game. We could just make a web game with JavaScript, or a pc game with some Java libraries. But overall the feeling i have for this project is: i'm super

hyped right now. I've always loved gaming and game developing, so getting assigned a project to develop a game is such a good thing for me. Also because we need to use make a web-game with Java back-end, we need to figure out how to use different tools and i think it's a really good learning opportunity. It also helps us combine the knowledge from different cpe classes we have learned before into a proper project.

Autoz: ส่วนตัวแล้วคิดว่าการได้ทำโปรเจคมันก็ดีนะ มันทำให้ได้เรียนรู้แบบชนิดที่เข้าใจจริงๆเพราะว่าต้องจมอยู่กับพวกมันนานมากกว่าจะแก้ปัญหาต่างๆนาๆได้ มันทำให้เราได้พบปัญหาใหม่ๆบางปัญหาที่เราไม่คิดว่ามันจะเกิดแต่มันก็เกิด แล้วการที่เราได้หาปัญหาต่างๆนั้นด้วยการ DEBUG ด้วยตัวเองมันทำให้ได้เรียนรู้อะไรหลายๆอย่างมากขึ้น ทำให้ได้เข้าใจบางเรื่องที่ไม่เข้าใจในตอนที่เรียน แต่พอมาทำแล้วก็เข้าใจก็มี

Nont: คิดว่าน่าจะมีปัญหาเรื่องการเขียนโค้ดแรกๆจะช้า พอรู้ละว่าตรงนี้ต้องมีอะไรทำยังไง หลังๆมาเริ่มเข้าใจงานของกลุ่มตัวเองละ พอจะรู้ว่าถ้าเกิดปัญหาตรงไหนและไปหาทางแก้ตรงไหน ยัง งง เรื่องของที่ต้องคุยกับ front-end อยู่บ้างชอบนะตอนได้ทำโปรเจคแต่ต้องรู้เรื่องด้วยถ้าไม่รู้เรื่องละไปทำคงไม่ค่อยสนุกเท่าไหร่ ทุกโปรเจคแม้จะทำไม่สำเร็จหรือยังไงอย่างน้อยเราก็ได้อะไรกลับมาอยู่ดี

11 Learning Resources

Here are the learning resources we used to learn the necessary knowledge to create this project.

- Spring https://www.youtube.com/watch?v=If1Lw4pLLEo
- Spring MVC https://www.youtube.com/watch?v=g2b-NbR48Jo
- Using Spring to create API https://www.youtube.com/watch?v=9SGDpanrc8U
- Build and Deploy Java Web : Spring Boot https://code.visualstudio.com/docs/java/java-webapp
- Spring and React - basic https://github.com/spring-guides/tut-react-and-spring-data-rest/tree/master/basic
- Spring and React - hypermedia https://github.com/spring-guides/tut-react-and-spring-data-rest/tree/master/hypermedia
- Spring @PostMapping @RequestBody w/example https://www.youtube.com/watch?v=nsQb5Au6EHQ
- Spring @CrossOrigin for CORS problem https://spring.io/guides/gs/rest-service-cors/
- Spring @Component & @Autowired https://www.youtube.com/watch?v=K43qyHJXmWI