CPE 200 Semester 2 2021
Project CAREN
Design Overview Document

Baangkok Vanijyananda, Ukrit Kosonsomboon, Nonthawat Kongsichai

Version of February 8, 2022

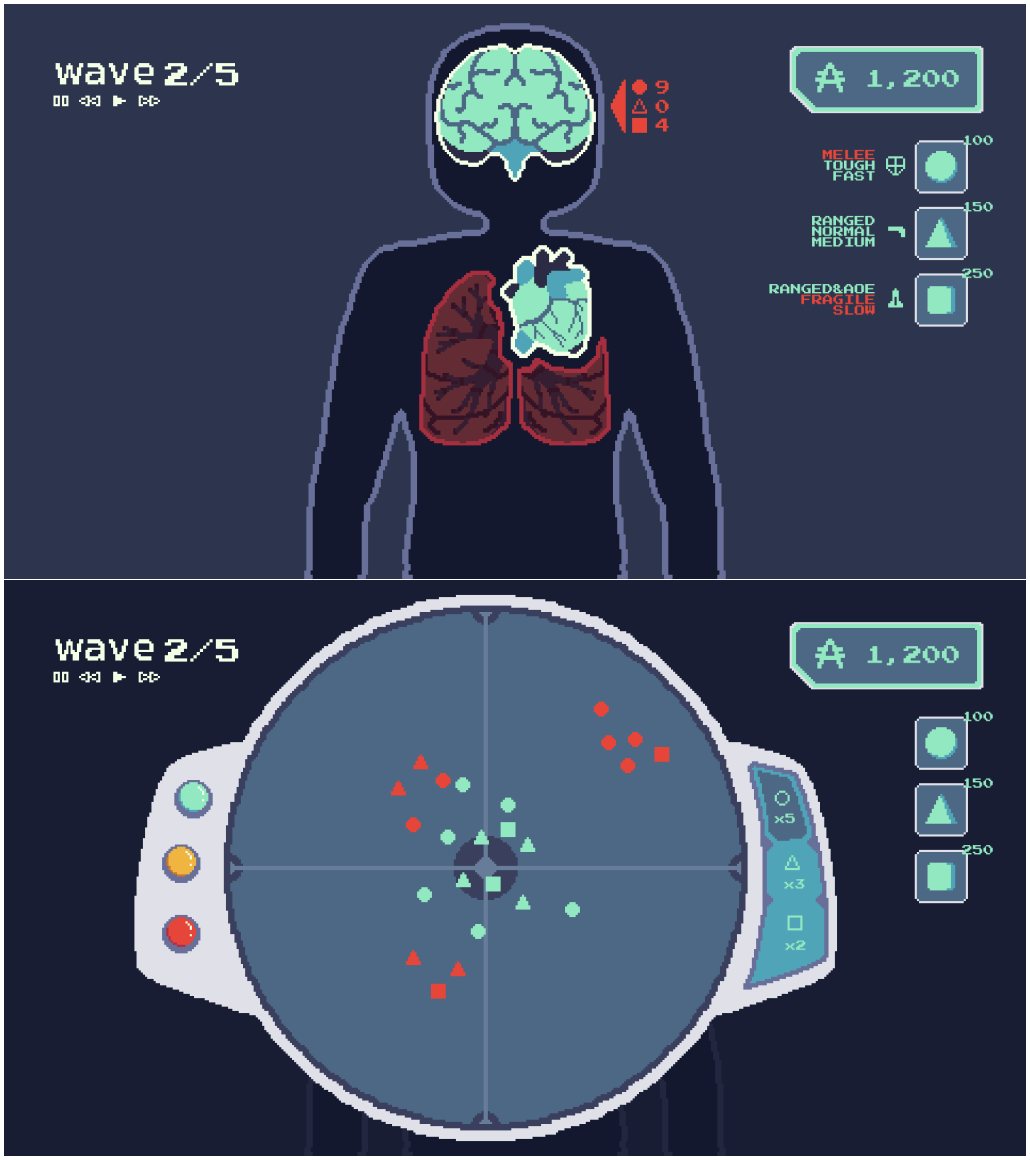
1 Introduction

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.

2 Game Overview

2.1 Figma Prototype

Figma Prototype Draft: CPE200-Project



2.2 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 at least one area survives through all the waves, the player survives the pandemic.

3 System Architecture

3.1 Model

- Speed Multiplier
- Current wave
  - Count
  - Target Area
  - Total virus count for this Wave
  - Virus type for this wave
- Zoom state
- 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)

3.2 Class Hierarchy

Interface are denoted with: >

Class are denoted with: -

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

3.2.1 Front-End

- TextObject
- ImageObject
- AudioObject

3.2.2 Back-End

- Game (Singleton)
- Shop (Singleton)
- Area
- Inventory (Singleton)
- WaveManager (Singleton)
- Wave
- TimeManager (Singleton)
- ZoomManager (Singleton)
- Tokenizer
- BehaviorEvaluator (Parser)
- GeneticCodeManager (Singleton)
> Node
  - ProgramNode
  - StatementNode
  - CommandNode
  - MoveCommandNode
  - AttackCommandNode
  - AssignmentStatementNode
  - BlockStatementNode
  - IfStatementNode
  - WhileStatementNode
  - NumberNode
  - VariableNode
  - RandomNumberNode
  - BinaryArithmeticNode
  - SensorNode
- UnitFactory (Factory)
- GameObject
  - Button
    - variationsOfButton (see *4.3 UML Diagram*)
    - Unit
      - Antibody
      - Virus

3.3 Tools

3.3.1 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: Back-end.

3.3.2 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 Detailed System Design

4.1 Key Classes and Interfaces

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

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.

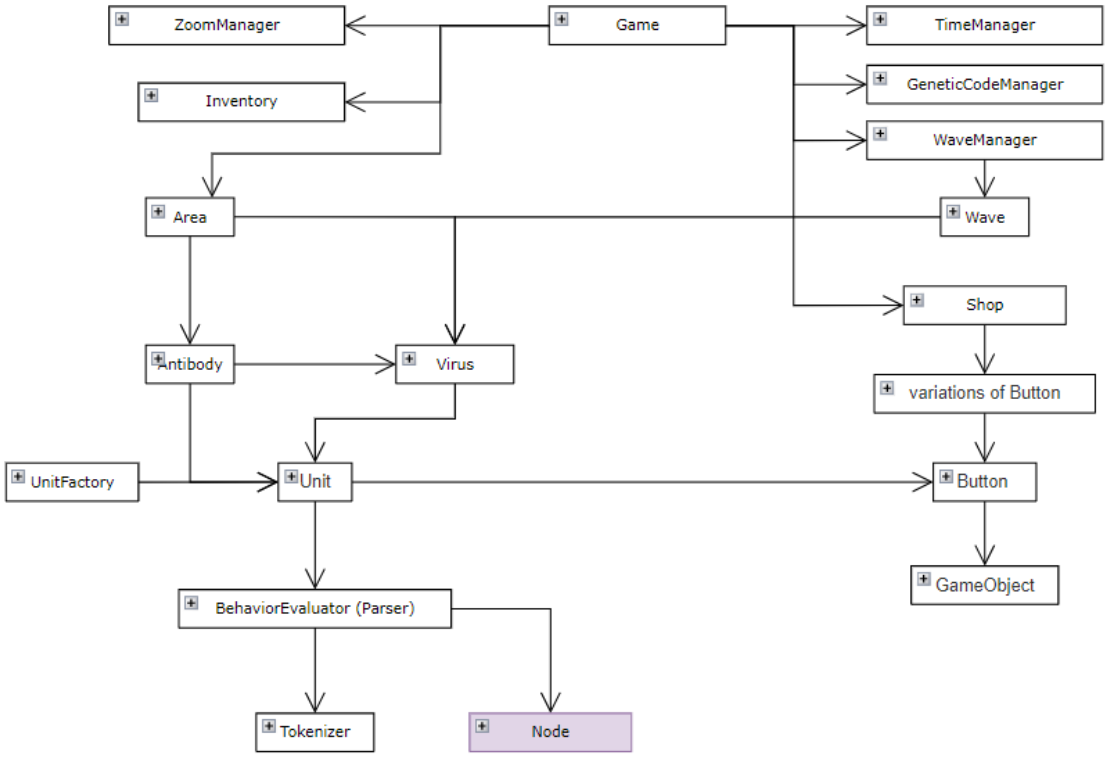
Node: The interface for all nodes that will be generated from the genetic code to create an AbstractSyntaxTree.

GameObject: The parent class for all things that should have a position.

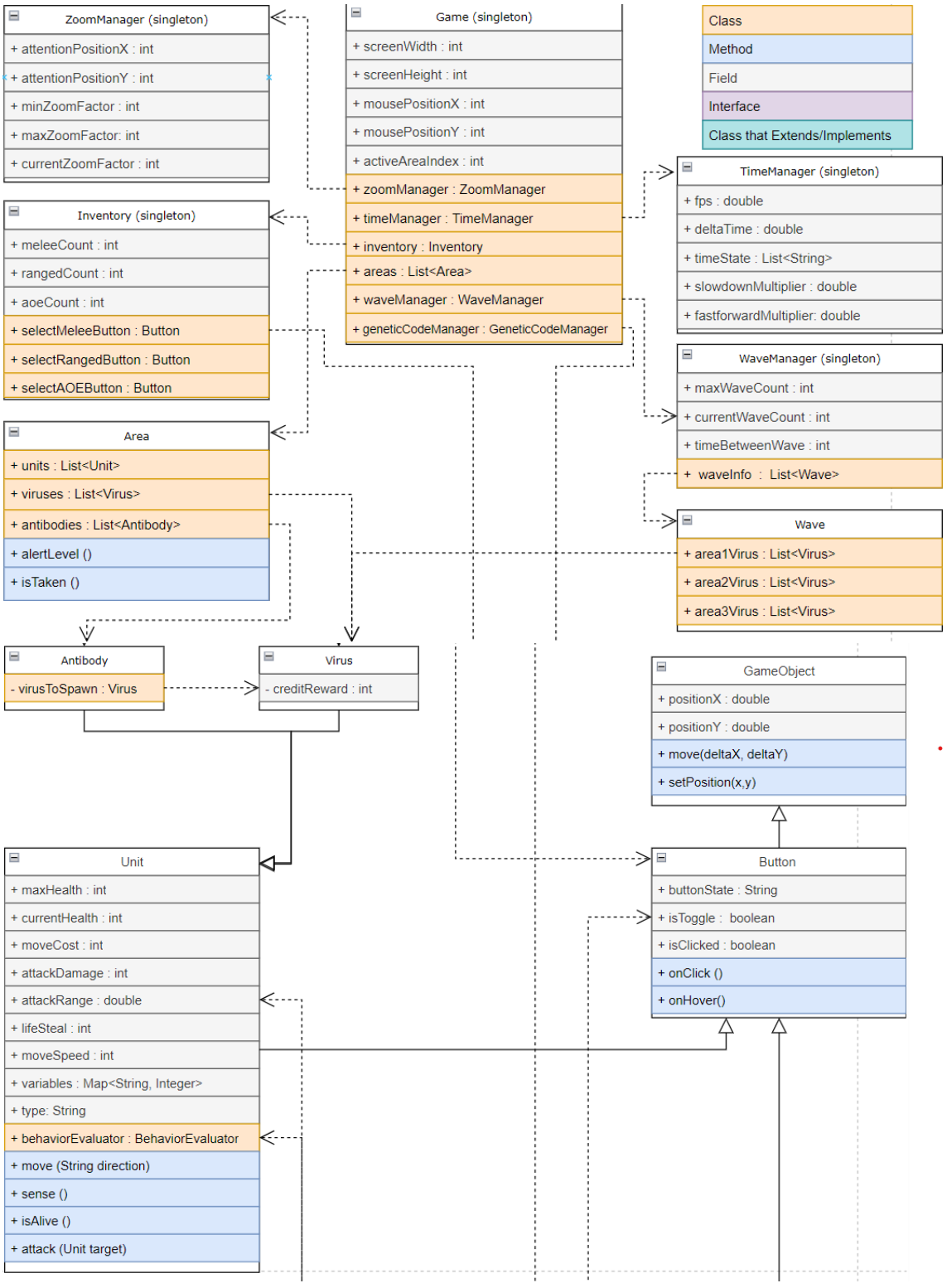
Button: The parent class for all things clickable.

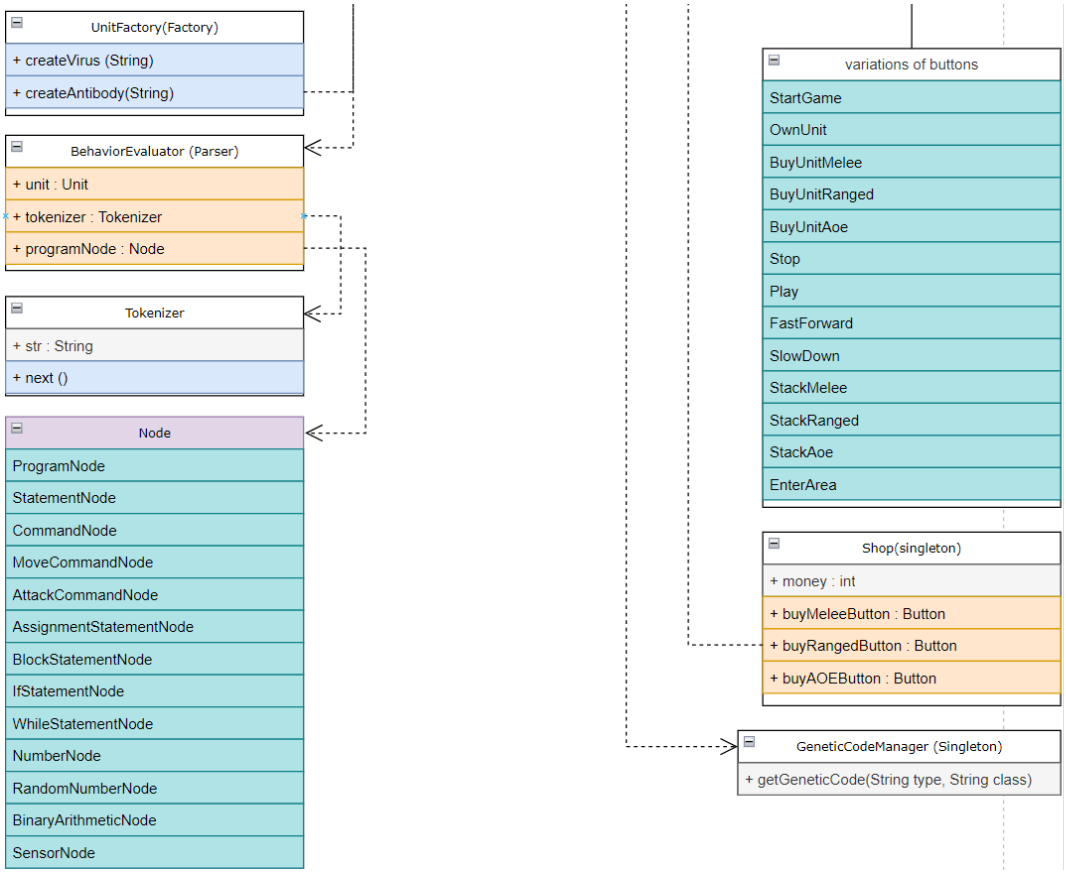
Unit: The parent class for Virus and Antibody.

4.2 Module Dependency Diagram



4.3 UML Diagram





4.4 Important Rep Invariant

Shop: Antibody credits must not exceed 9,999.

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.

GeneticCodeManager: Genetic code cannot be changed while playing.

GameObject: Position must remain within playable position.

Button: State must be between 0 to 4.

Unit: Current health must not exceed max health.

4.5 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.

4.6 Design Patterns

Factory Method: There are many Unit types with different stats. The UnitFactory 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.

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.

5 Testing

Testing plan

- Using JUnit to test methods
  - Game

    before and during coding : ก่อนที่จะรันเกมก็ต้องทดสอบก่อนว่าสิ่งที่ออกแบบมานั้นมันสามารถเอามารันจริงได้หรือปล่าว

    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)

- GameObject

  before and during coding : GameObject ที่กำลังจะเขียนขึ้นมานั้นสามารถนำไปใช้เป็น Game Object ได้จริงหรือไม่

  after and during coding : หลังจากที่เขียนเสร็จแล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าวแล้วลองนำไปใช้จริงๆดู แล้วดูผลลัพธ์ว่าสามารถใช้ได้จริงหรือไม่

- Button

  before and during coding : Button ที่กำลังจะเขียนขึ้นมานั้นสามารถใช้เป็นปุ่มในเกมได้จริงๆหรือไม่ สามารถกดได้จริงๆหรือไม่

  after and during coding : หลังจากที่เขียนเสร็จแล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าวแล้วลองนำไปใช้จริงๆดู แล้วดูผลลัพธ์ว่าสามารถใช้ได้จริงหรือไม่

  - variationsOfButton

    before and during coding : variationsOfButton(ปุ่มต่างๆในเกม) ที่กำลังจะเขียนขึ้นมานั้นสามารถใช้ได้จริงตามหน้าที่ของแต่ละปุ่มหรือไม่ เช่น ปุ่ม Buy กดแล้วก็ซื้อของ เงินลด , ปุ่ม Back กดแล้วกลับไปหน้าก่อนหน้านี้ เป็นต้น สามารถกดปุ่มที่ไม่สามารถกดได้หรือไม่ (เช่นเงินไม่พอ แล้วลองกดปุ่ม Buy แล้วจะเกิดอะไรขึ้น)

    after and during coding : หลังจากที่เขียนเสร็จแล้วยังสามารถทำให้ test case ที่คิดไว้ก่อนหน้านี้เป็นจริงได้หรือปล่าวแล้วก็ดูว่ามีบัคใหม่ๆเกิดขึ้นรึปล่าวแล้วลองนำไปใช้จริงๆดู แล้วดูผลลัพธ์ว่าแต่ละปุ่ม

สามารถทำตามหน้าที่ของมันได้จริงหรือไม่

- 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

Coverage Analyzation

คิดว่า Testing plan ที่เราเขียนมานั้นค่อนข้างที่จะครอบคลุมพอสมควร แต่ว่าสิ่งที่ขาดไปคงจะเป็นเรื่องรายละเอียดบางส่วนที่เราเองยังไม่ค่อยแน่ใจเท่าไหร่ เช่น เราจะ test class นี้ กระบวนการไหน ที่เขียนไปก็ยังมีแค่ before กับ after เท่านั้น

6 Workplan

6.1 Work Allocation

Further details will be added in the future.

1 **Designing and Design Overview Document**: All members
2 **Front-end React, JavaScript**: Member#1
3 **Back-end Parser, Parser related class**: Member#2
4 **Back-end Game, GameObject, Button, Unit**: Member#3
5 **Back-end Other classes**: Member#1,2,3

6.2 Work Schedule

We use Trello to manage our work schedule.

1 **Designing and Design Overview Document**: by Feb 8
2 **Front-end React, JavaScript**: by Feb 23
3 **Back-end Parser, Parser related class**: by Feb 12
4 **Back-end Game, GameObject, Button, Unit**: by Feb 16
5 **Back-end Other classes**: by Feb 23

Further details will be added in the future.

6.3 Key Dependency

Key Dependencies are Parser and Parser related class, Game, GameObject, Button, 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.

Front-end mostly will have lower load of works since all the graphical assets are already done. Bottle neck will happen in back-end development. So we arrange everyone to work on that after Feb 16. With exception that Member#1 who will need to continue working on the front-end until the project is finished, he gets lower back-end work load.

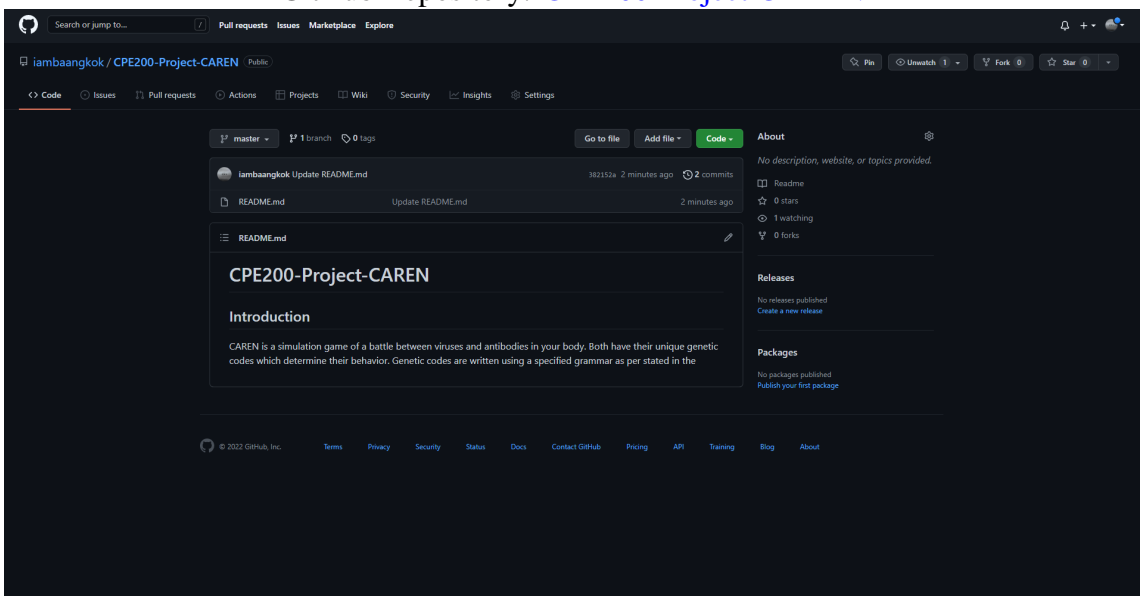
7 Known Problems

– 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 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.

8 GitHub Repository

GitHub Repository: CPE200 Project CAREN



9 Comments

Time spent on the design: Around 40 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.

10 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