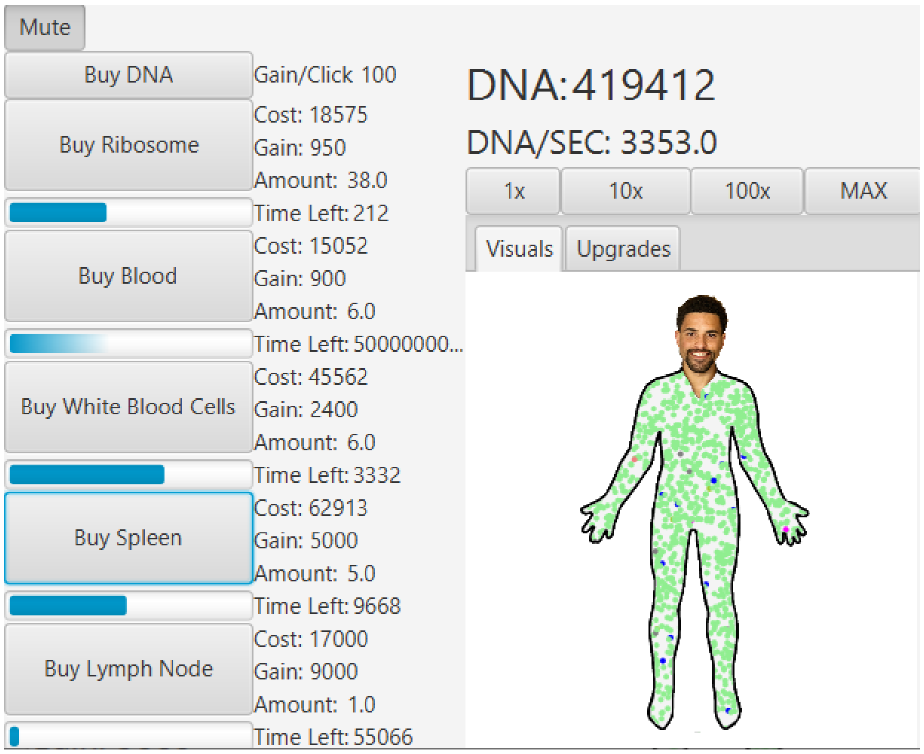
COVID Clicker

Design Manual

An incremental game created by
Brillitte Orozco
James Howe
Joseph Carluccio
Michael Gertz



Gain/Click 100

Cost: 18575
Gain: 950
Amount: 38.0
Time Left: 212

Cost: 15052
Gain: 900
Amount: 6.0
Time Left: 50000000...

Cost: 45562
Gain: 2400
Amount: 6.0
Time Left: 3332

Cost: 62913
Gain: 5000
Amount: 5.0
Time Left: 9668

Cost: 17000
Gain: 9000
Amount: 1.0
Time Left: 55066

DNA:419412
DNA/SEC: 3353.0

Introduction

On a high level our game follows in the footsteps of other basic Incremental/Idle/Clicker(I will be using these terms interchangeably from here on) but mainly in the same vein of Adventure Capitalist, where the main structure of the game consists of what we've been calling "producers". Producers can be bought with the main resource, and when bought they create *x* amount of that resource(the gain) every *y* seconds. Buying more of the same producer adds the *x* to the amount gained and increases the price by some multiple, buying a certain number of producers decreases *y* by some amount. We also have a button which allows the user to gain a set amount of the resource by clicking it.

The aesthetic we decided to use for our game was the human immune system, so each of our producers was named something related to the immune system, and the resource they were creating was DNA.

For our game we used a modified MVC framework, where the model and the controller were normal java files, but the view was handled by a FXML file created by SceneBuilder. Otherwise we used the basic conventions of the framework, where the model holds the current state of the game and the information pertaining to it, while the controller handles the actual control of the game, and linking the information in the model to the view.

Our model holds the basic information regarding the game as javaFX properties, and holds all the information relating to the producers as a list of producer objects, which each hold the information specific to each producer.

Our producer object holds the information relating to each producer, such as the gain per period, the period(amount of time between getting the gain), the cost, and how these values should be increased when more are bought. Besides storing the information, it also has a variety of methods for interacting with the producer, such as what to do when one is bought.
Each producer runs it's own thread to keep track of its own time and will update what's needed each time it counts to zero.

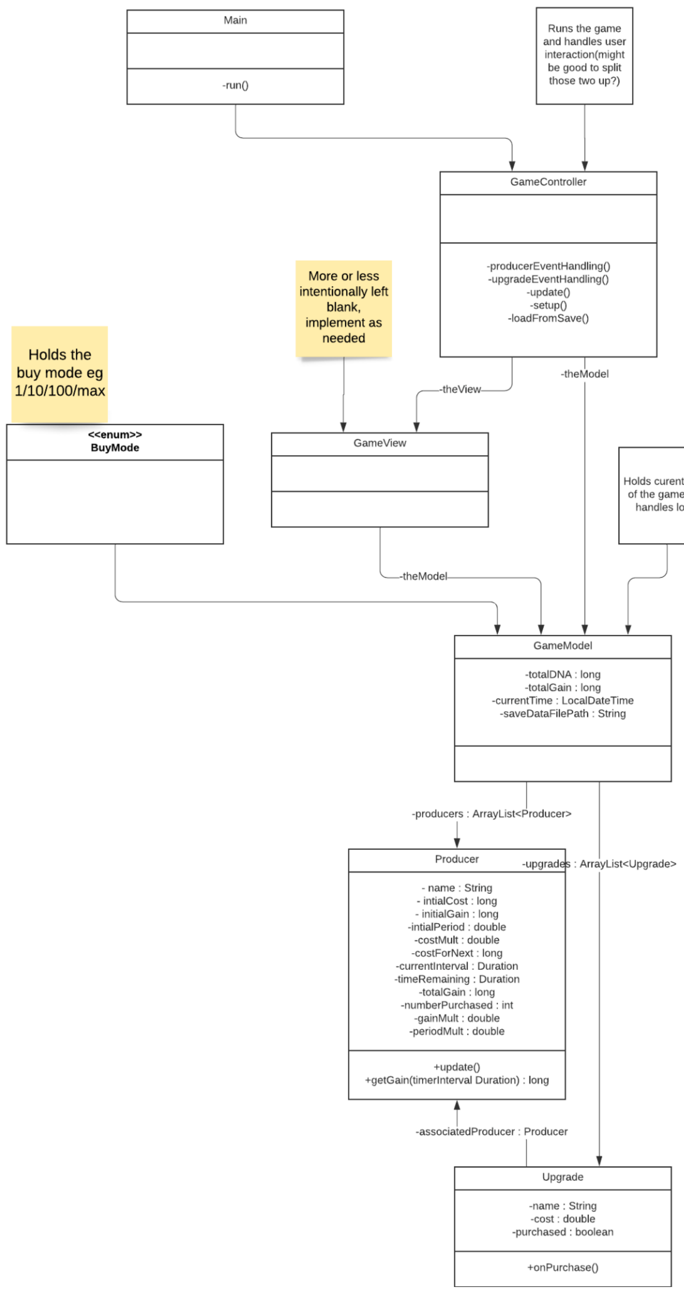
User Stories

Our user stories were fairly successful, with the majority of them being completed in some form or another. The ones which were not able to be implemented were abandoned fairly quickly as we were developing the scope of our game. The priority we assigned turned out to be fairly different then the priority of a developer; the order in which these were completed doesn't line up to the order they are in here.

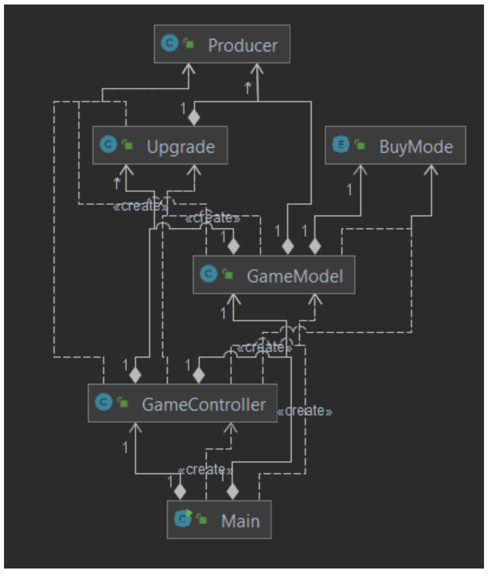
User Stories	Priority	Completed?
As a user, I'd like to be able to start the game	5	Yes
As a user, I'd like to be able to buy cells	5	Yes
As a user, I'd like to be able to see the game	5	Yes
As a user, I'd like to be able to see my current amount of DNA	4	Yes
As a user I'd like to be able to buy more cells	4	Yes
As a user I'd like to be able to see how much it costs to buy things	4	Yes
As a user I'd like to see if my mouse is hovering over something	4	Yes
As a user I'd like to be able to see my dna/sec	4	Yes
As a user I'd like to be able to change between fullscreen and window	3	No
As a user I'd like to be able to determine if the game hasn't been opened for a bit	3	No
As a user I'd like to be able to see my stats	3	No
As a user, I'd like to be able to buy cells by 1x/10x/100x/max	3	Yes
As a user I'd like to see some visual representation of the immune system	3	Yes
As a user I'd like to be able to upgrade my cells	2	Yes
As a user I'd like to see particle effects when something is clicked	2	Kinda
As a user I'd like to be able to mute the game	1	Yes

Design

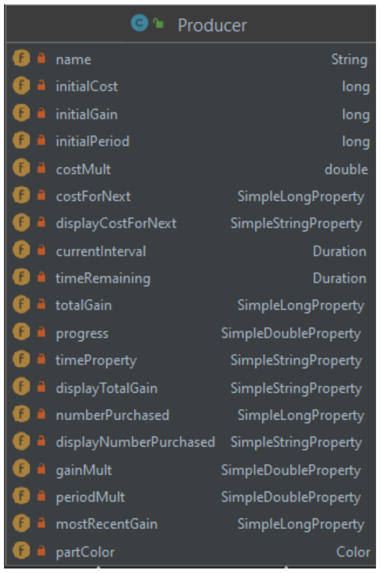
Our original UML diagram covered the general hierarchy of the program, as briefly described in the introduction. There are some parts to this which ended up not being used, but for the most part the general structure is in place.



We don’t have any CRC cards since they were never talked about in class, and weren’t mentioned in any of the client meetings, while the UML diagram was.



Our actual UML diagram follows the same general structure as our intended structure, with a bit more interdependency between the objects. The majority of the main methods from the original were implemented in our final design.



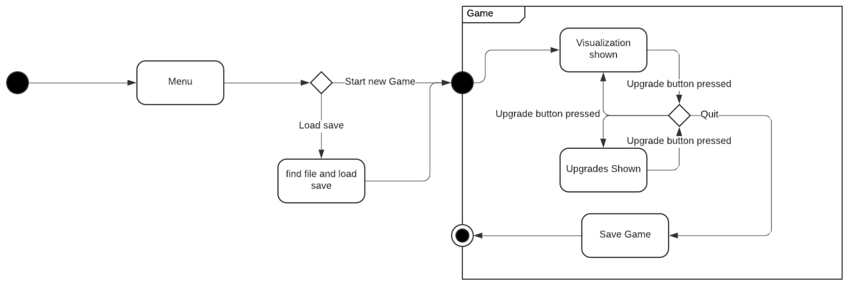
Producer	
name	String
initialCost	long
initialGain	long
initialPeriod	long
costMult	double
costForNext	SimpleLongProperty
displayCostForNext	SimpleStringProperty
currentInterval	Duration
timeRemaining	Duration
totalGain	SimpleLongProperty
progress	SimpleDoubleProperty
timeProperty	SimpleStringProperty
displayTotalGain	SimpleStringProperty
numberPurchased	SimpleLongProperty
displayNumberPurchased	SimpleStringProperty
gainMult	SimpleDoubleProperty
periodMult	SimpleDoubleProperty
mostRecentGain	SimpleLongProperty
partColor	Color

Our producer class stores the most information out of all of our classes having to store pretty much all the information relating to the game except for total DNA

The controller class is mainly split into two types of functions, those which setup game objects through binding them to information from the model, and event handlers which perform actions on model when certain events happen in the scene(such as a button being pushed). If given more time I would like to rework Controller into two different classes which handle these separately, since there is a stark divide between them.

Throughout the design Javafx bindings and change listeners were used extensively to not only link the model to the view, but to link values of different types in the same class. For example in producer the string values which are displayed in the view are bound to the numerical values used in the producer. Since each producer also used a thread since they each had to keep track of a timer independently of each other, Javafx change listeners were invaluable for keeping track of the current state of each producer and linking it to the view. This of course connects to the user story “being able to see the game”

One large aspect to the user stories which we weren’t able to get to, was any functionality relating to saving the game since it just fell outside of our range. Along with this, we were also not able to implement the state diagram we originally intended for the game(shown below) since its main purpose was for handling saves and loads of the game. The Game loop was implemented more or less as shown except we the upgrades and visuals button separate.


Menu
Start new Game
Load save
find file and load save
Game
Visualization shown
Upgrade button pressed
Upgrade button pressed
Quit
Upgrade button pressed
Upgrades Shown
Save Game