Java Graphics Report

The Ecosystem Project

SUBMISSION AND SETUP
I have submitted:
- Zipped folder containing the project at various stages of development
- PDF report
- Video of program running
- Final code

To run the program, please download Processing https://processing.org/download/ Then navigate to the final code folder, and place the folder called ‘Ecosystem’ into the Processing folder (it might be called libraries on other operating systems, but it is called Processing on OS), select all files and right click ‘open’. This should automatically open the pde files in processing. Then simply hit the run button.

CONCEPT
I wanted to do something using evolutionary algorithms, and visualise this process. I have decided to use Processing, because it affords a great deal of control over everything and I am interested in building all the mechanics and interactions myself.

I decided to make a micro ecosystem styled to look like a petri dish. Two main species interact, the prey (which eat plants) and the predators (which eat the prey). The prey will be given behaviors to seek out and feed on plants, while the predators will hunt the prey. Over time, many creatures will die off, but some successful creatures will have the opportunity to pass on their ‘genetic’ material to offspring. The idea is to see some evolutionary changes (size, speed,

turning ratio) over several generations to better match their individual goals, gather resources, and survive.
To mimic reproduction and reward more successful creatures, every collision between two same-type creatures has a small possibility of generating offspring. The genetic material of both parents (such as size, turning percentage, or max speed) are combined as a range. These numbers are further randomised through a small change of mutation (if the mutation rate is too large, it will undue to value of selective reproduction).

CHALLENGES

Creating an ecosystem where everything does not just die off quickly is difficult, and early deaths would prevent evolution. To this end, it was important to establish a sort of balance early.

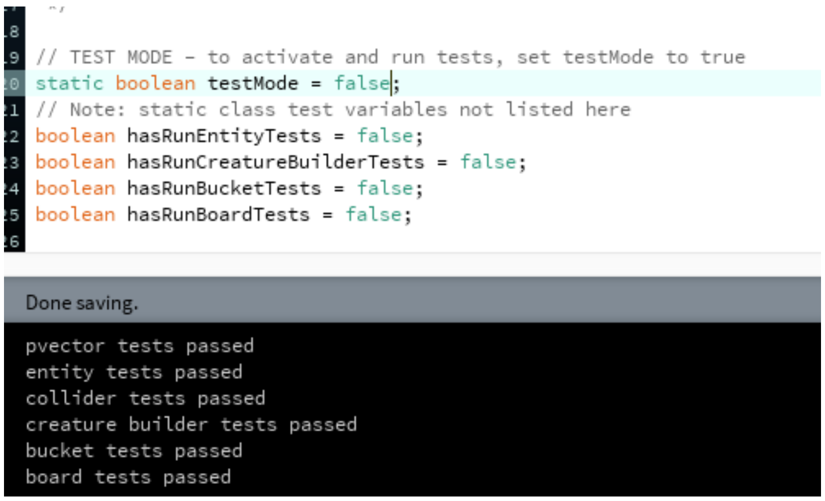
As I worked and played with the numbers, I found a few useful parameters to help maintain a balance. For plants, I had them reproduce on a rough timer (they reproduce faster if the plant population falls below a minimum value). Further, to create clumps, I made it so newer plants form close to their parent. I looked to nature to help balance the prey-predator dynamic. The starting prey population is much larger than the starting predator population. Since prey are much more likely to collide at first, each reproduction produces one offspring. For the predators, as in nature (such as lions and wolves) each reproduction can produce up to three offspring, as predators collide much less frequently than prey.

DESIGN

I used the MVC pattern from the start, and also the Component Architecture System. All classes were designed and connected together to maintain these two patterns. Any changes to the graphics would be contained within the Render class. Due to the nature of processing, there were a few constraints placed on handling user input, and the event listeners (for key presses) need to be located in the Ecosystem class, rather than an input class.

TESTING

Processing imposed a few limitations. Processing does not allow for a main method in classes, this is because when it converts the pde file to a java file, it does so as one massive file rather than individual classes. To get around this, I created a static boolean hasRunTests, so that for each class, the tests are run only once. I also created a global static boolean testMode, so that tests may be turned off in the main program. For some classes, I created special constructors to be used within the tests for simplicity and control. Every method and class that has complex (or sometimes simple) logic underwent automatic testing, and of course unit testing for the classes.



```
// TEST MODE - to activate and run tests, set testMode to true
static boolean testMode = false;
// Note: static class test variables not listed here
boolean hasRunEntityTests = false;
boolean hasRunCreatureBuilderTests = false;
boolean hasRunBucketTests = false;
boolean hasRunBoardTests = false;
```

```
Done saving.
pvector tests passed
entity tests passed
collider tests passed
creature builder tests passed
bucket tests passed
board tests passed
```

For the submission, I have turned off testing, but it can be easily turned on by changing the topmost variable in the Ecosystem class - testMode - to true. Tests run as each class is constructed for the first time, so some tests print as passed in the first screen, and the rest after the user interface slider screen (after which the entities and game board are actually built).

PLAN (STAGES OF DEVELOPMENT)

To help structure the project and clearly lay out my goals, I have broken the work down into 10 stages. In the ‘Stages’ folder I have included all work throughout development. I have also included a video of the program running.

STAGE 1: Basic Structure

Create a simple entity that moves around an empty space, its health runs down and it dies with time. Find a visual way to represent health. Creature cannot leave the arena (screen). Build a simple physics system.

To start, I wanted to have a clear MVC structure, and started with a separate render class to handle all graphical aspects. I created my basic classes to such as a vector class, entity class, transform class, and render class. From my previous experience, I chose to avoid inheritance and instead used a component architecture system. Thus, the entity has several components, such as a transform (to hold position, scale, and heading), and a render component (to handle all graphics aspects).

I applied random forces to the entity to make it move to get started. I chose to visually represent health using the alpha channel (it fades as health diminishes).

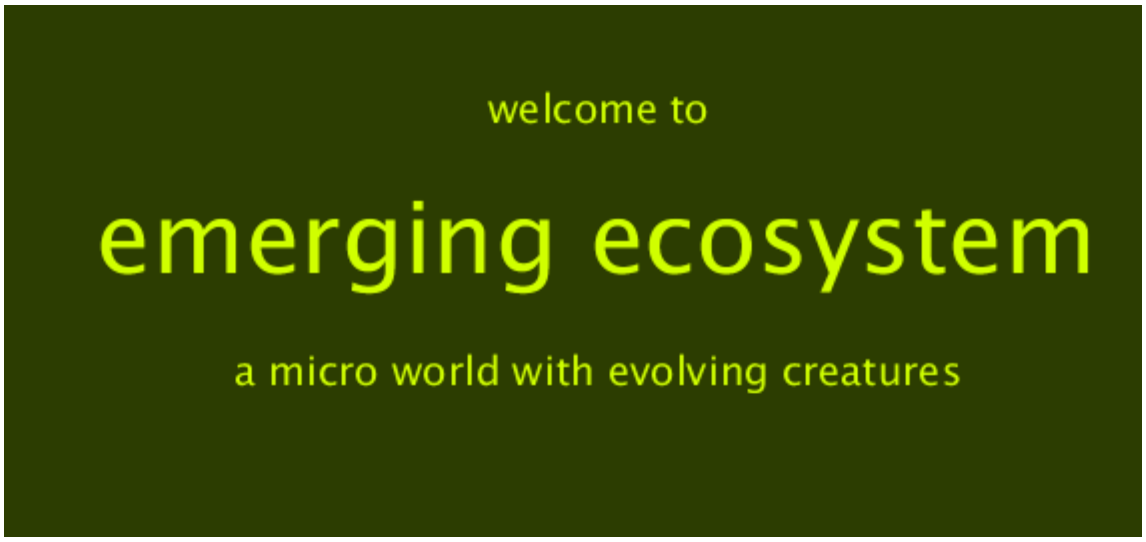
STAGE 2: Life and Death

Next, I refactored so that when the prey creature dies, the screen changes. To do this, I created a loop in processing’s main update function that checks the gameScreen variable (this section of the code serves as my game manager). This also makes it possible to easily add in more screens, such as a title screen, a user interface (UI) screen, and a game over screen.

I then wanted to have several creatures at once. To make this manageable, I made the creature part of an array of entities, and add several creatures to it. The simple array list of entities could then be looped over and updated. Later on, this array could be used to check for collisions.

I chose a simple color scheme that recalls nature and life, light green-yellows and darker greens. For the entities, I chose colours to allow them to contrast each other and stand out from the background. Red for the predators seemed intuitive, blue for the prey, and various shades of green (randomly determined) for the plants.

Start screen:

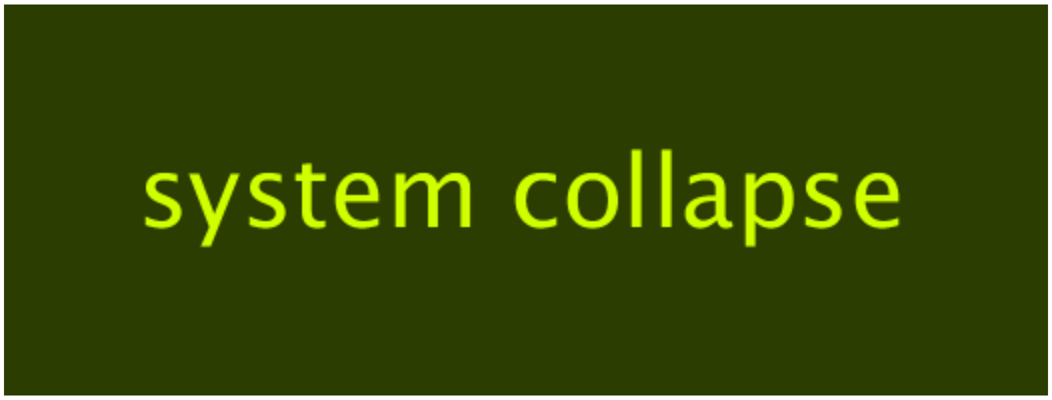


welcome to

emerging ecosystem

a micro world with evolving creatures

Game over screen:



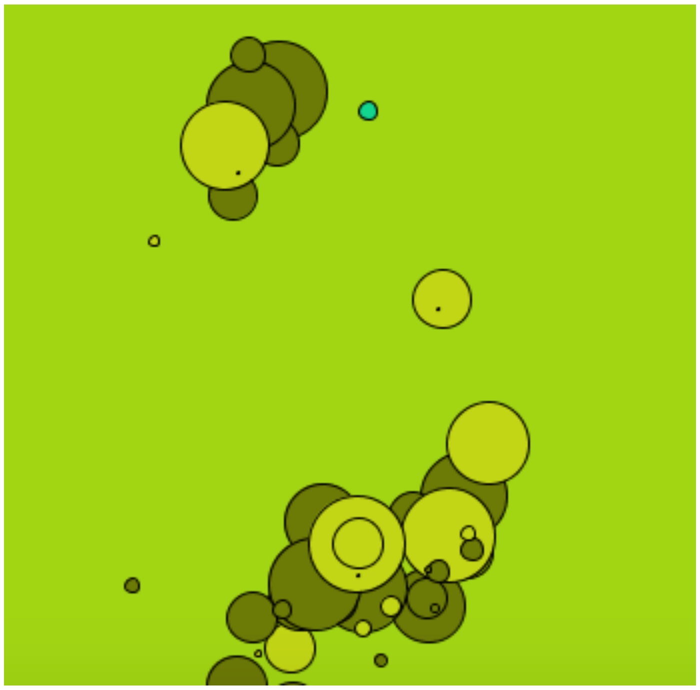
system collapse

STAGE 3: Adding Plants and Refactoring
At the start, randomly spawn “plants” that are distributed around the arena in clumps or in different formations - these serve as a food source for the prey creatures.

At the start of the ecosystem, a random distribution of plants are scattered around the arena. Plant entities do not lose health or die, unless they are eaten. I needed to refactor the project to separate my entity array into different lists, one for plants and one for prey entities. This allowed me to reduce the number of collision checks for optimisation, because I do not need to check if plants are colliding. I then added a static collider class to handle all collisions, using the entities vector location and scale to determine overlap. Once the collision check was working on prey and plants, I needed to do another minor refactoring of my main game loop to handle working with multiple prey and plants.

After collisions were set up, it was time to add plant growth. My first attempt spiralled out of control and slowed down the program. I solved this by limited growth more severely (adding in a sort of plant timer), and a min and max plant population limits. Next I wanted plants to grow near other plants, to form clumps of green and open areas without food to make the arena more interesting and challenging for the prey. I randomized plant colour for some visual variety.

I then refactored again, to return to using a single creature list, as it is more manageable, and makes it easier to add and delete prey once predators are added. To avoid collision detections on all entities, I first perform checks on the tags (Tag.PREY for example) and only check collisions on those entities that I care about. I further optimise by removing double collision checks, by limiting the second for loop to be less than the first.

The image below shows how plants grow in expanding 'clumps', leaving empty desert areas for creatures to navigate.


STAGE 4: Basic Reproduction
Add ability of prey to reproduce on collision (there is no genetic material, just clones for now)

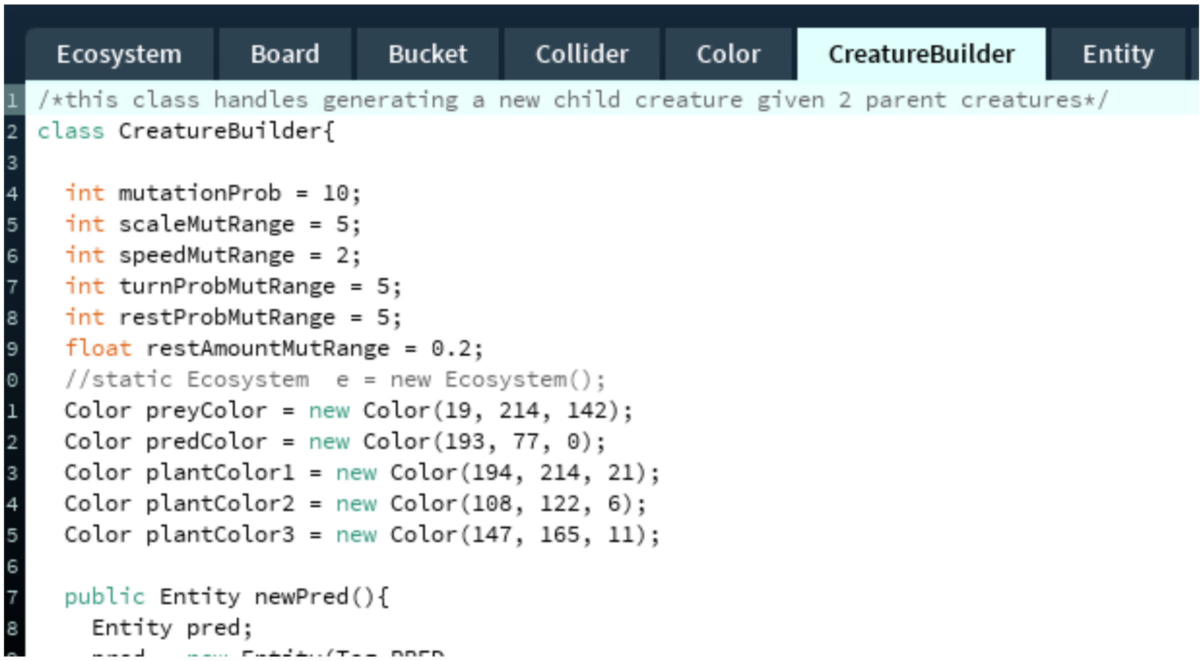
I decided that a simple way to handle offspring is to have there be some possibility of a new creature created whenever two entities of the same type collide. This way, entities that survive longer will have the potential to generate more offspring and pass on their genes.

At first I simply added newly ‘born’ creatures to the entity list right away, but this ended up generating exponential growth and crashing. To avoid this, I created a new Array ‘birthList” and only added newly born creatures to my entity list at the end of every update cycle, which prevents newly born creatures colliding with the parent creatures and creating more births.

STAGE 5: Genetic Material and Evolution
Add genetic component and a builder class to generate a new creature given the two parent creatures

I started simply at first, with an entity’s variable parameters as size. To make the genetic factors more meaningful, I refactored my physics system to take into account mass (determined by scale). Thus, the larger an entity is, the slower it is too. Once the system was working, I added more inheritable parameters, such as the likelihood an entity will turn, to what degree it will turn, how often and for how long it will ‘pause’ in place and so on.

These parameters were then wrapped up in a Creature Builder class (a factory) to allow for the generation of many different types of entities. The creature builder class is also responsible for births, in which it takes in two entities of the same type, and combines their genetic material for the offspring. Thus, if entity A is size 5, and entity B is size 10, the offspring has a random chance of being anywhere within the range of 5 to 10. I add in mutation as well, so that once the offspring’s size has been determined (say 7) a mutation has a small chance of occuring and reducing or increasing this number by a small or large amount, to create more variety in the gene pool. Helper classes, such as color, were also written to clean up the code.



```
/*this class handles generating a new child creature given 2 parent creatures*/
class CreatureBuilder{

  int mutationProb = 10;
  int scaleMutRange = 5;
  int speedMutRange = 2;
  int turnProbMutRange = 5;
  int restProbMutRange = 5;
  float restAmountMutRange = 0.2;
  //static Ecosystem  e = new Ecosystem();
  Color preyColor = new Color(19, 214, 142);
  Color predColor = new Color(193, 77, 0);
  Color plantColor1 = new Color(194, 214, 21);
  Color plantColor2 = new Color(108, 122, 6);
  Color plantColor3 = new Color(147, 165, 11);

  public Entity newPred(){
    Entity pred;
```

STAGE 6: Adding Predators

Add a system that allows creatures of type PRED to eat the PREY and gain health. At first, all entity movement is random, both prey and predators. Predators and prey also have the same chance of reproduction at first, but this leads to system collapse quickly.

This stage required refactoring the main game loop to include another entity type (predators) as well as further optimization to the collision check (as this is the area of the greatest computational requirements).

STAGE 7: Creature Behavior
Refactor creature behavior - want the random movements replaced with a variety of search algorithms for the prey (to locate food) and hunting algorithms for pred to track and follow prey.

I changed the randomised movement to straight movement, with a random chance of turning some amount on each turn. This turning probability is inheritable. To counter this, I also added a rest probability to allow creatures to pause or slow down occasionally. This trait could be useful for prey, because their food source grows in clumps.

To further improve the behavior, each predator entity is given a pointer to a current prey to hunt down and eat (unless it dies on its own first). Once the predator no longer has an active prey to hunt, it receives a new random prey to chase.

To see if I could encourage more stationary prey, I refactor how the plants work, such that plants give +10 every update a prey is there, until they die (they have 100 food points total). More complex behavior may be added to the prey in the future (such as stopping them completely if they are feeding) but the system works for now so I will continue as is.

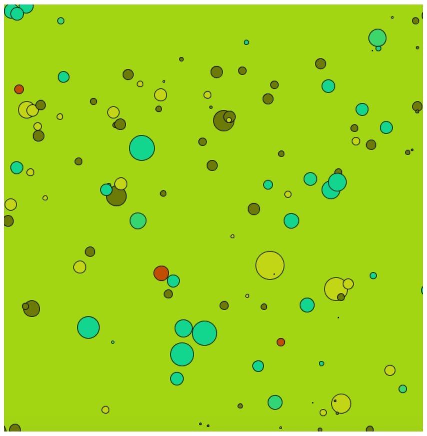
STAGE 8: Space Partitioning Algorithm and Refactor
Optimise collision checking.

I wanted to implement a space partitioning algorithm. To do this I created a board class and a bucket class, where a board is a collection of buckets and each bucket holds a variable number of entities. Then I compare collisions only within buckets, except dealing with edge cases specially.

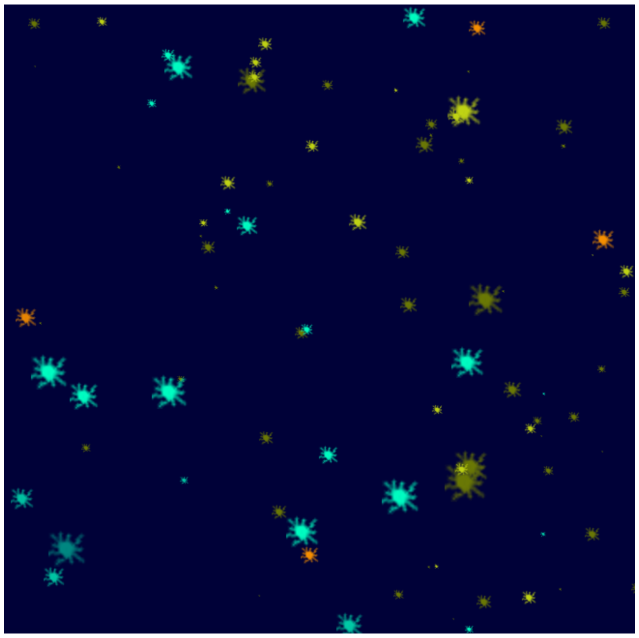
This change results in a major refactor, to migrate most of the collision and main game loop to the new board class. Optimisations are that far fewer entities are checked for collisions, speeding up performance. Within each bucket, I optimise further to avoid double looping over the array (keeping the inner loop smaller than the outer loop), avoid collision checking non-consequential entities by Tag, and prevent an entity collision checking itself.

STAGE 9: Improved Graphics Refactor - From Images to Animations
Initially, I used simple circles to represent each creature and plants. The alpha channel was controlled by the health amount, so creatures faded as they lost health. Color information is

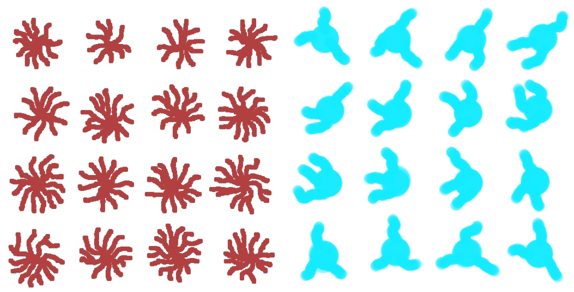
carried by each creature, so that the colors can be modified with the evolutionary algorithm in the future if desired.



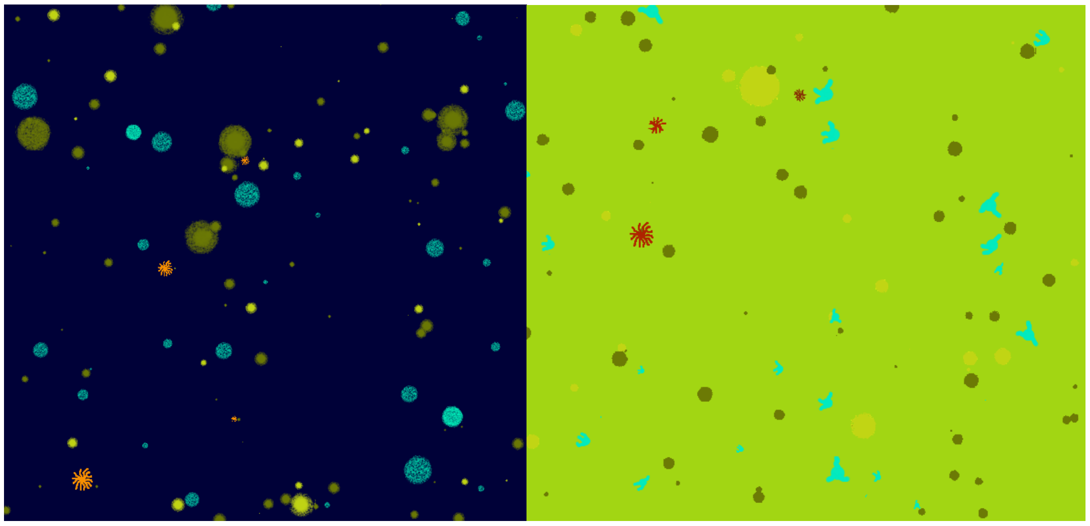
Next, I wanted to attempt adding in images. At first, I loaded the images in my render component. This resulted in a very slow program, because the images were being loaded individually for each creature, which was a waste of resources. I refactored the project to load the images in the set-up phase, and then pass pointers to these images to the render components when an entity is built by the entity builder class. This gave equal performance to the simple shapes. I used a single image, and added different tints, to preserve the original colors and the health-alpha channel relationship.



The next step was to add animated sprites, to give more ‘life’ to the creatures. To create the sprites, I used the open source software Gimp and drew custom sprite sheets. The images below show the predator (red) and prey (blue) sprite sheets. Since the entities are meant to be simple lifeforms in a petri dish, the styles were kept simple, with appendages to allow for ‘swimming’ across the screen.

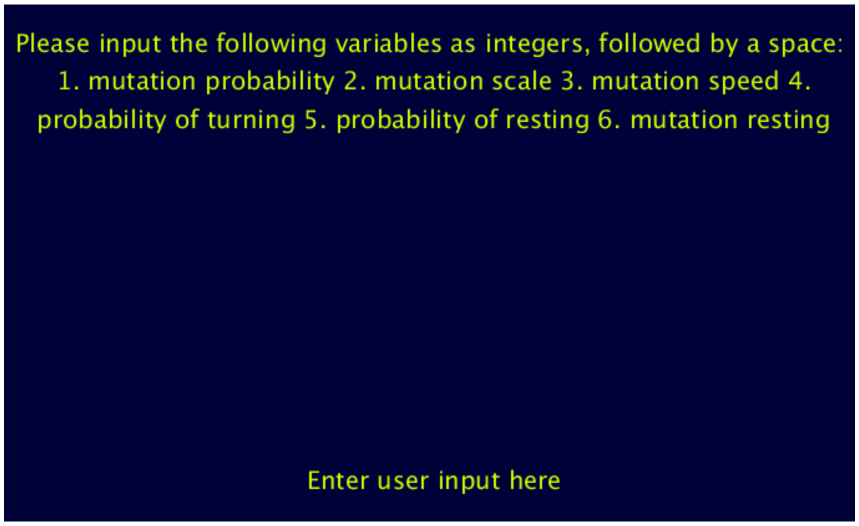

Custom sprite sheets created with Gimp.

The first challenge was to find a way to keep track of all the sprite information (x, y position of the current sprite, for example) so I decided to add this information to the renderer. Alternatively, I could create a new Sprite component and handle the information there, but I chose to put it directly on the render component because I would just be passing the sprite component to the render component anyway, which might add unnecessary complexity.

The image below is a screen shot from the animated sprites. For now, the sprites sheets provide motion (waving arms for the predators, shifting texture for the prey) in the style of a petri dish. Later additions could be that the heading determines the sprite rotation, so small creatures could be made to move purposefully about the screen.

I then randomized the starting sprite index, to prevent the animations all happening at the same time in step with each other. I have included a short video of the program in my submission.

STAGE 10: User Interface
Finally, I wanted to make the program more interactive through a user interface in which users can input the starting probabilities of the ecosystem.

My first approach to the UI was to have a string that can be manipulated by the user, such that deleting or typing updates the string they see on the screen. Then, when the user presses ‘ENTER’, the string is parsed by a helper method, and the variable values (separated by spaces as indicated in the instructions to the user) will be fed into the variables such as mutationProbability, turnRate etc so the user may impact the outcome of the ecosystem directly. The image below shows the first attempt at a UI screen.

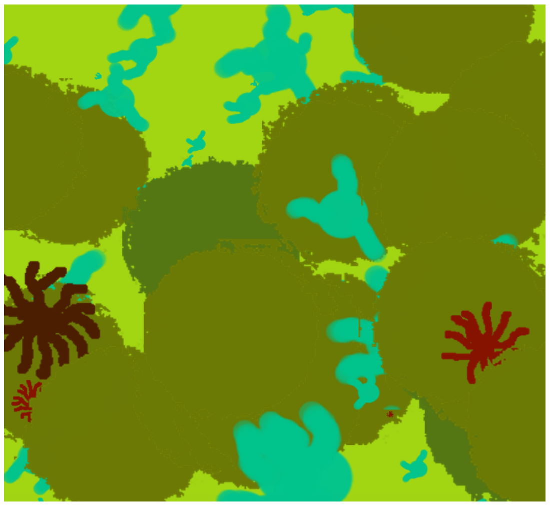


Please input the following variables as integers, followed by a space: 1. mutation probability 2. mutation scale 3. mutation speed 4. probability of turning 5. probability of resting 6. mutation resting

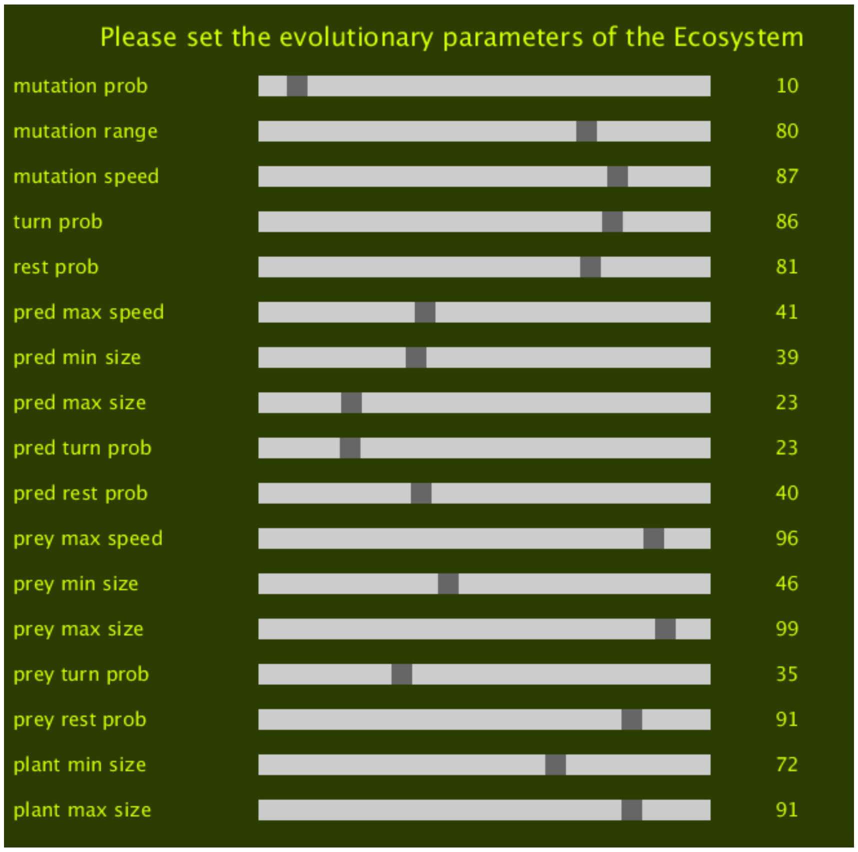
Enter user input here

This method works, but visually it could be clearer for the user. Also, the user could enter bad information (such as letters rather than numbers, or decimals) and break the program. So I refactored to use sliders instead, and added a Scrollbar class to handle the scroll value logic, and an event listener for the enter key. This also required refactoring a bit in the main screen, and setting up arrays of strings for the slider names, arrays for the slider values, and arrays for the sliders themselves. This helped me deal with the large amount of sliders and information effortlessly with for loops.

Once I got the sliders working and generating the rules for the ecosystem and creature creation, the next step was to limit the slider ranges to appropriate ranges, to avoid the extremes (such as creatures as large as the screen, as in the image below). I also added in measures to deal with minimum values being larger than maximum values (in such cases, the ecosystem reverts to a default minimum of 1).



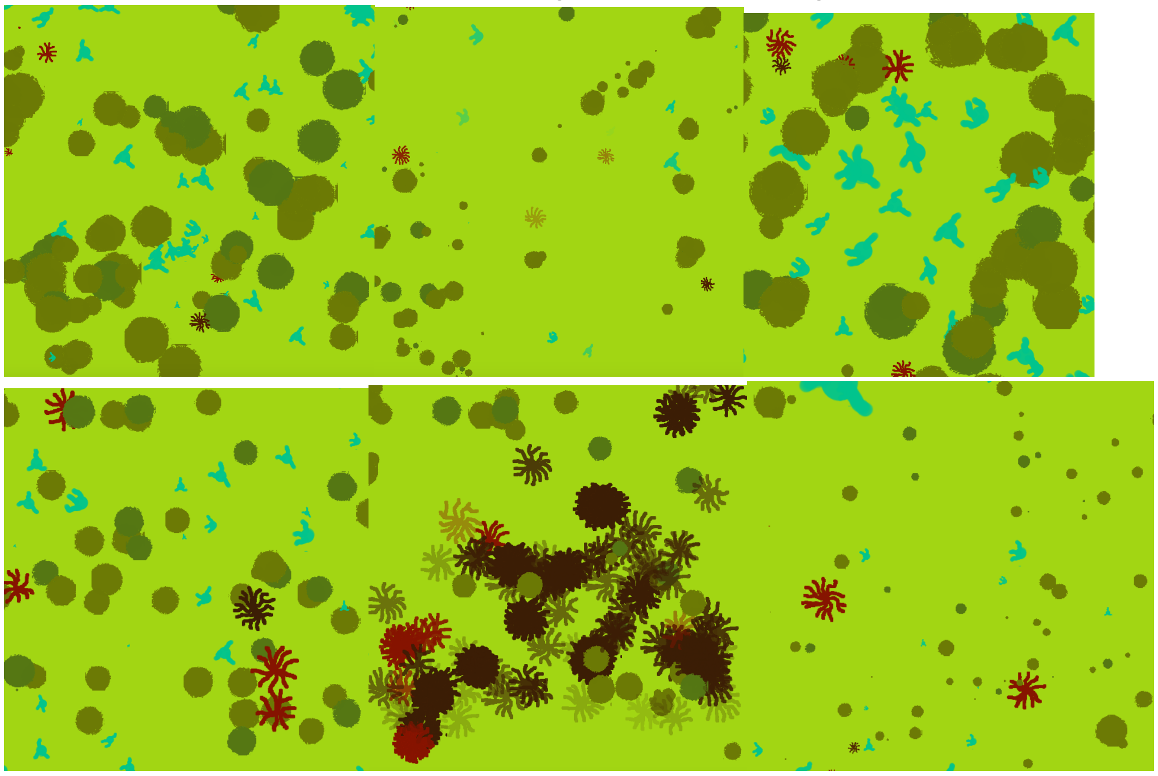
With the updated constraints, the inputs are controlled and the input experience should be straightforward for the user.

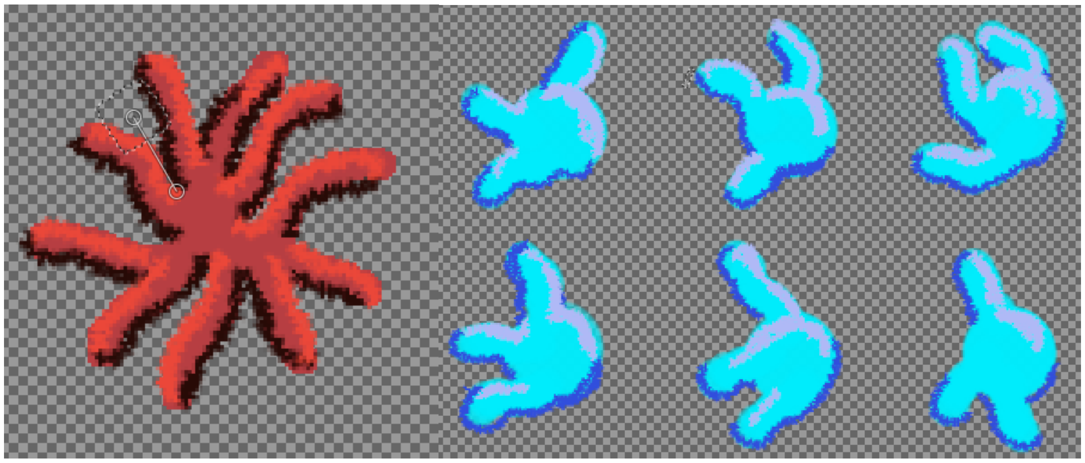


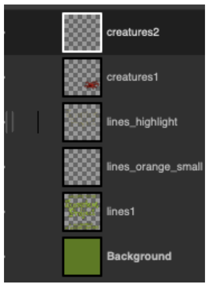
Please set the evolutionary parameters of the Ecosystem

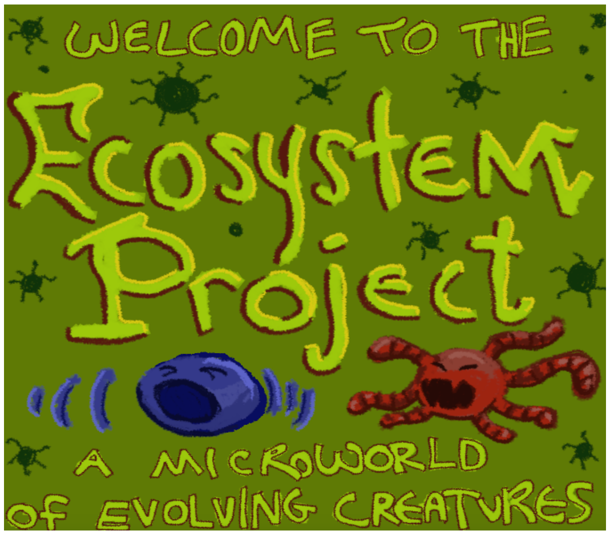
mutation prob	10
mutation range	80
mutation speed	87
turn prob	86
rest prob	81
pred max speed	41
pred min size	39
pred max size	23
pred turn prob	23
pred rest prob	40
prey max speed	96
prey min size	46
prey max size	99
prey turn prob	35
prey rest prob	91
plant min size	72
plant max size	91

The variable parameters allow the user to generate a wide range of possible outcomes.



STAGE 11: Further Graphics
I decided to improve the graphics further, by adding improving to my entities using Gimp:

To improve the entities, I added shadows and highlights to give the shapes more definition, and make the animations more entertaining. In doing so, I kept to my original color schemes.

Next, I added both an introductory screen (all custom made in Gimp, using layers and other advanced features such as opacity, shading and curves).


I created both a welcome screen and game over screens using Gimp:


CLOSING REMARKS

I added in a few final touches, such as a maximum allowable prey and predator count, to prevent a population explosion that slows down the program or halts it. Now, the program runs smoothly every time. I also added in the option to restart by hitting the ENTER key after reaching a game over, or at any time during the program to restart and set different ecosystem parameters.

GOING FORWARD

I am happy with the project as it is, and believe it is a fully contained project that reached a clear endpoint. I also passed some of my stretch goals for the project. If I had more time (or perhaps later on my own) I also want to create a chromosome that codes for a specific search algorithm, so that hunting strategies can also be inherited. I would also like to automate running the ecosystem with a large variety of start parameters, to determine which combination of parameters generate the most stable (longest lasting) ecosystem.