

A-Level Computer Science NEA Project

EVSIM

Table of Contents

1.0 Analysis	7
1.1 Computational Methods.....	7
1.1.1 Abstraction/concurrence	7
1.1.2 Decomposition/selection	8
1.1.3 Data Mining	8
1.2 Stakeholders	8
1.3 Research on existing solutions	9
1.3.1 Evolution Simulator by Primer Learning.....	9
1.3.2 Evolution Simulator by Cary Huang	11
1.4 Essential Features	14
1.5 Limitations	16
1.6 Hardware and software	17
1.6.1 Hardware Requirements.....	17
1.6.2 Software Requirements	18
1.7 Success Criteria	18
2.0 Design	20
2.1 Problem Decomposition.....	20
2.2 Structure of the solution	21
2.3 Algorithm Design	23
2.3.1 <code>MapGenerator.java</code> Map Generation.....	23
2.3.2 Organism generation and movement.....	24
2.3.3 Organism attributes	26
2.3.4 Terrain Attributes.....	31
2.4 Usability Features	32
2.5 Variables and Validation	33
2.5.1 <code>MapGenerator.java</code>	33
2.5.2 Organism Generation and movement	34
2.5.3 Organism Attributes.....	36
2.5.4 Terrain Attributes	38
2.6 Iterative Test Data.....	39
2.7 Post-development Test Data.....	41
3.0 Development	43
3.1 Front End UI.....	43
3.1.1 Display of the Map.....	43
3.1.2 Opening Menu.....	48

3.1.3 Display of the organisms.....	52
3.1.4 Start game options menu	61
3.1.5 Main menu options	80
3.2 Calculations and processing	91
3.2.1 Time	91
3.2.2 Organism Movement.....	95
3.2.3 Organism Evolution.....	97
3.2.4 Displaying stats to the user.....	101
3.3 Organism Statistics	108
3.3.1 Awareness	108
3.3.2 Food	113
3.3.3 Size.....	123
3.3.4 Speed	126
3.3.5 Temperature	128
3.3.6 Preferred Habitat.....	129
3.3.7 Health	132
3.3.8 Survivability	135
3.4 Organism Movement.....	136
3.4.1 Logic.....	136
3.4.2 Code.....	136
3.4.3 Testing.....	137
3.4.4 Testing survivability	137
3.5 Organism Breeding	138
3.5.1 Breeding.....	138
3.5.2 Evolutionary adaptations.....	151
3.5.3 Implementing evolutionary adaptations	151
3.6 Adapted organism movement.....	156
3.6.1 Correcting the diagonal bias.....	156
3.6.2 Organism movement based on other organisms	157
3.6.3 Testing the adapted organism movement.....	159
3.7 Predator / Prey Cycles	160
3.7.1 Logic.....	160
3.7.2 Allowing organisms to become predators.....	160
3.7.3 Testing if the organisms can become a predator.....	161
3.7.4 Organisms attacking	162
3.7.5 Testing the attacks	163

3.7.6 Revisioning the attacking method	163
3.7.8 Testing the new attacks	164
3.8 Saving and Loading Simulations	166
3.8.1 Saving the JSON	166
3.8.2 Implementing the ability to save	167
3.8.3 Testing the saved JSON	169
3.8.4 Second Implementation	171
3.8.5 Testing the second implementation	172
3.8.6 Loading the JSON Logic.....	173
3.8.7 Implementing the ability to load the JSON.....	173
3.8.8 Testing the loaded JSON	177
3.9 Working on the user experience	179
3.9.1 Organism appearance.....	179
3.9.2 Side menu information.....	180
3.9.3 Allowing the user to prevent predator/prey cycles.....	181
3.9.4 Console log	182
3.9.5 Balancing evolution	184
3.9.6 Map preview.....	187
3.10 Natural Disasters / Seasons	188
3.10.1 Logic.....	189
3.10.2 Implementation	189
3.10.3 Testing resultant temperature change	191
3.11 Packaging EvSim and getting user feedback.....	192
3.11.1 Testing the packaged project.....	192
3.11.2 Stakeholder feedback	193
3.11.2.1 Options Menu Bug.....	193
3.11.2.2 Eastings and Northings	196
4.0 Post development testing.....	198
4.1 Testing for function.....	198
4.2 Testing for usability.....	202
5.0 Evaluation	204
5.1 Comparing success criteria and test data.....	204
5.1.1 Allowing the user to change and set attributes during the creation of the simulation	204
5.1.2 Manipulation and ability to save currently processing formats	205
5.1.3 Displaying organism attributes	206
5.1.4 Creating and displaying a sufficient world which has fluid biomes.....	207

5.1.5 Demonstrating evolution physically and visually.....	208
5.1.6 Calculating and storing changing attributes	209
5.1.7 Operating with changing parameters.....	209
5.1.8 Taking inputs from the user during execution of simulation.....	209
5.1.9 Demonstrating evolution over ‘generations’	210
5.2 Assessing Usability.....	212
5.2.1 Opening menu	212
5.2.2 Colourblind accessibility	213
5.2.3 Side menu.....	214
5.3 Maintenance and Limitations.....	215
5.3.1 Limitations	215
5.3.2 Maintenance.....	216
5.4 Future Improvements.....	217
5.4.1 Organism movement	217
5.4.2 Changing attributes during execution	218
5.4.3 Improving the appearance	218
5.4.4 Increasing the scope of evolution.....	219
CODE REFERENCES.....	220
C1 Processing.....	220
C1.1 Main.java	220
C1.2 OpeningMenu.java	220
C1.3 StartGameMenu.java.....	223
C1.4 LoadSimulation.java	226
C1.5 MainMenuOptions.java	229
C1.6 GameWindow.java.....	236
C1.7 GenerateMap.java	240
C1.8 GenerateOrganisms.java	242
C1.9 PerlinNoise.java	245
C1.10 Organisms.java	247
C1.11 SideMenu.java	264
C1.12 SaveSimulation.java.....	269
C1.13 LoadSimulation.java	271
C1.14 RenameFile.java.....	274
C1.15 ErrorWindow.java.....	276
C2 Models.....	276
C2.1 Colours.java	276

C2.2 Options.java.....	277
C2.3 Organism.java	278
C2.4 Settings.java.....	279
C2.5 Stats.java.....	280
C2.6 LoadSimulation.form	284
C2.7 MainMenuOptions.form.....	286
C2.8 RenameFile.form	288
C2.9 SaveSimulation.form.....	290
C2.10 SideMenu.form.....	292
C2.11 StartGameMenu.form	299
C3 Packages	303
C3.1 Terrains	303
C3.2 FoodTypes.....	305
C4 Storage.....	310
C4.1 options.json	310
C4.2 organisms.json.....	311

1.0 Analysis

One of the many wonders of the world is how we ourselves and other animals have evolved overtime. While not studying biology personally, I have contacts within the subject base with people who have expressed their interest in software which shows evolutionary development of organisms' overtime depending on environmental and genetic factors. There is no program currently in place that demonstrates both criteria, and level of depth successfully. As such, it is my belief that a program like this could be extremely beneficial to aspiring biologists and perhaps those already established in the field.

My solution would be to develop a program which would randomly generate a global model within which environmental conditions and population based factors are considered for the development of constantly adapting organisms. Based on where the organism is, their evolutionary traits will develop differently. The evolutions will take place over a simulated timescale, with each organism being affected by their environment, as well as other local organisms. Overtime, by the end of the iteration process, organisms should gradually gather in concentrated positions which best suit them and their offspring.

1.1 Computational Methods

The solution to the problem identified in 1.0, would be suitable for a computational approach due to the amount of processing and calculations that need to take place during the iteration of the program. Probabilities and multiple influences need to be calculated effectively over a very small period, much faster than any human could potentially do. Furthermore, a computational approach provides a visual demonstration of the theory and practice of evolution, therefore providing a more accurate representation of evolution, and meeting the needs of the stakeholders' success criteria.

1.1.1 Abstraction/concurrence

Abstraction within this program is going to be a key factor due to the intensive processing that is needed. The program must calculate not only the probabilities and the independent factors on each organism concurrently, but the control of each organism through AI must also be executed. This AI system will be working concurrently with the processing to find the best evolutionary gains (different areas of the map). This *emphasises the need for abstraction within the program*, as well as *concurrence*. Abstracted elements of the program include those such as the attributes used during the movement of organisms, and how they are represented to the user. The organisms will move depending on their environment around them, using information about the environment and the organisms around. Logically, an organism should not be able to move from one side of the map to the other, therefore when deciding where to move, only the area around an organism should be checked, instead of checking the entire map.

Since the problem at hand requires a program which allows the user to *view* the evolutionary stages over a long period of time, advanced 3d graphical visualisation is simply not required, as a simple 2d object on a map would be able to sufficiently visualise each organism. Furthermore, there is an abundance of 2D shapes that are available to use, which can be used to demonstrate the difference in evolutionary stages of organisms. For example, an organism which can evolve for water may be visualised as a circle, compared to a land mammal being a square.

- This would provide the end user with enough *visual* information to be able to tell the difference between organisms, as well as each one's individual stats (see 1.4).

In the larger picture, the accuracy of EvSim itself should be addressed. Evidently, the theory behind evolution is incredibly complex, a process which would take years to perfect and replicate in code. Furthermore, the aim and purpose of EvSim, is to demonstrate evolution to those who are learning about the theory of evolution. Therefore, one of the largest abstractions in EvSim is the intricacy of evolution itself. So long as the program displays organisms adapting to their environment, then evolution is displayed, and the aim of the program is met. This therefore means that there are going to be some

attributes which lack the depth of others (for example food), as they are not necessarily needed to achieve the wider aim of EvSim.

1.1.2 Decomposition/selection

Decomposition with EvSim is going to play a significant role in both the *code optimisation* and the *development* stages. This is due to the extremely large array of probable ‘evolutions’ that can be developed.

- Using decomposition to break down both the creation of the code and the simulation itself would not only be useful for the creation of code, but also debugging and optimising to allow the process to run in complete.

For example, when it comes to creating the process of evolution itself, I will breakdown the stages, for example allowing the organism to eat before other attributes are created which require the food contribution. Attributes like speed and size depend on what food an organism eats, therefore food will be developed before size and speed, however both will contribute evolution individually.

Combining this with *selection* allows for a smarter approach to processing, by removing the unnecessary code depending on the environment (for example no organisms in one area – don’t calculate the probabilities) we can significantly reduce the demand created for processing.

- The execution of EvSim is expected to be considerably expensive on system resources, due to the amount of attributes and organisms to update each time the program iterates. Selecting the attributes to update for each organism means that the program is not executing code which has no use. For example a dead organism will not need its attributes updating anymore, therefore it can be skipped from processing, saving on system resources and creating a better user experience overall.

1.1.3 Data Mining

Data mining will be of great use during the *decomposition* and *selection phase* due to the fact it will find common trends between different evolutionary chains. Links between adaptations will develop a ‘common’ chain of evolution and therefore selection can take place, with other chains of evolution being omitted from processing. For example, if an organism starts in the water, they will not progress down a desert chain of evolution for example.

- Data mining also reduces the amount of work that is placed on decomposition, due to its ability to simplify down larger problems into smaller trends of data. Furthermore, data mining (throughout iterations) can reveal more processes that can be abstracted, and therefore a simpler path of decomposition.

-The use of data mining in EvSim will allow the program to recognise trends in organisms and their parents to give them more specific adaptations. For example, if many generations of parents to one organism share common attributes, then the child organism will have a large benefit to that specific attribute.

1.2 Stakeholders

Stakeholders for EvSim would be current/past/present biologist students and/or teachers. After speaking to both students and teachers who are currently studying/teaching biology at A-Level, they have stated interest in software that would achieve what EvSim aims to do. This is primarily for the ability to test against hypotheses. Stakeholders have also however stated their interest in customisation the program, for example changing probabilities, from the creation of the world to the effects of the environment etc. An expression of interest in the use of time within the simulator is also key, with the ability to change the speed of simulation, as well as pause, save, and export data that is currently being used by the program.

- This would allow the user to compare the organisms at the start of their cycle to the end/middle. Data that would be provided by this program would be beneficial to conduct experiments with hypotheses, the use case of the stakeholder.

The current use case for the desired market would be for the testing and experimenting with hypotheses and would be used around 1-2 times a week maximum. An example of a hypotheses that would be given as a result of EvSim would be; "Organisms that dwell within a water / ocean based environment will on average be larger in size due to the negligible effects of gravity compared to land based animals" They have expressed interest in having the system available on a website, however due to limited hosting availability and amount of processing required, it was also acceptable for the system to be available through a desktop app.

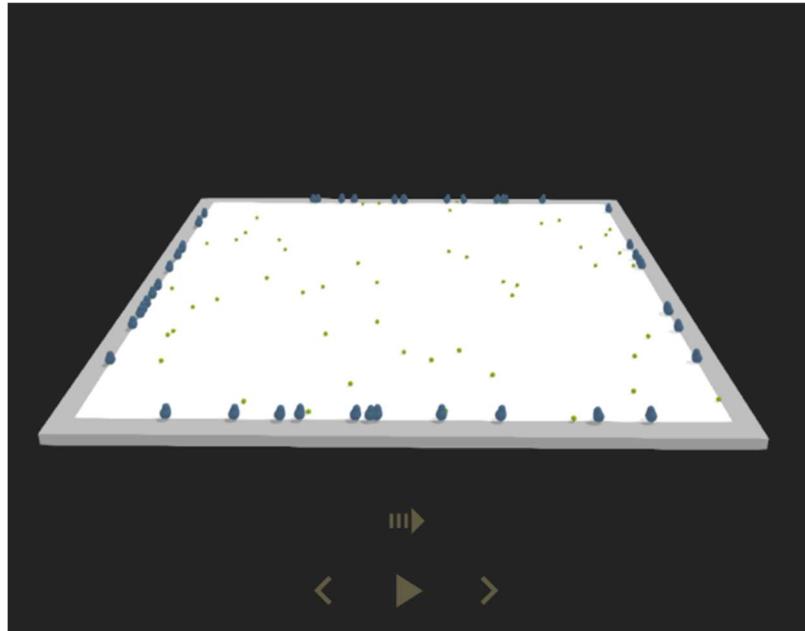
After conferring with the current stakeholders during an interview, the main focuses of this software, would be the ability to test and create different scenarios within the application. This therefore must be the focus, and with a lack of requirement for a complex GUI and graphics simulation, a 2d visualisation would be suitable for the user. Linking back to 1.1, the graphics side of this application would be abstracted, due to the little need and the little impact it has on the result of the stakeholder. Furthermore, due to hardware constraints, the lack of a complex GUI and graphical visualisation would be better for the product for stakeholders, as the program can focus on processing instead of visual output.

1.3 Research on existing solutions

After having a look around for current simulations/programs that aim to explore the concept of evolution, there are some examples that fit the criteria, however they are significantly lacking in features that explore scientific concepts (such as environmental and situational factors), rather they look at evolution in a linear format (e.g., organisms adapt over iterations to become better at one thing).

1.3.1 Evolution Simulator by Primer Learning

The Evolution Simulator by MinuteLabs (concept by Primer Learning), is the closest solution I have found to the current problem. This app is similar to my concept for the solution, primarily due to the fact it creates a 3d environment, with adapting organisms, that aim to survive through adaptations like speed and sense range. The Evolution Simulator in this example also can generate over 150 'generations' of organisms, with each one being adapted depending on how it survived in the previous iteration. Figure 1 (right) shows this program during the first iteration.



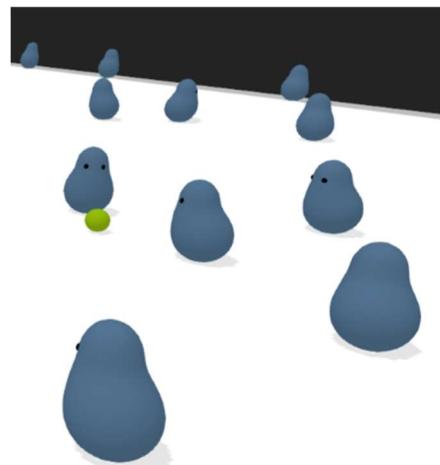
In this example, there is a blank 3d plane, where there are 'organisms' which adapt depending on their survival rates. Each organism has to receive a green object, and return to the corner. Over each generation/iteration of the program, the organisms develop one of 7 statistics (age of death, speed, sense etc) depending on their previous success.

This example of an evolution simulator is similar yet vastly different to the project here. Listed below is the features that I would like to use and the features that will not be on my project.

Features I would like to use from The Evolution Simulator:

The ability to see what each organism is currently doing:

Including a chart of the general spread of each characteristic out of the population



- This feature I will be incorporating into my project, due to the fact it achieves one of the major aims - being able to view each organism and see what their individual traits are as well as what they are currently doing. This would not only develop the user experience, but would also allow for a more in depth analysis of the effects of the environment, making it easier to test hypotheses; one of the criterias from the stakeholders

The ability to pause, save, and move between generations of organisms throughout time.

- This feature allows the user to do more in depth analysis of the situation as it evolves, for example being able to pause to check out individual organisms aims, as well as the ability to change between evolutions and generations in order to see the difference. I will be using this feature on my project due to how much it improves the user experience and the ability to analyse each organism individually. A program which runs once and does not allow for user interpretation and analysis would not be ideal for the stakeholder to carry out hypotheses testing and therefore the project would not sufficiently solve the problem at hand. This was also one of the criteria set by the stakeholders in 1.2.

Features I would like to omit/change from The Evolution Simulator:

The factors affecting organisms:

- It appears that this project has focussed heavily on their graphical design, (see images above), with a 3d world of organisms and food. This has come at a cost of an in depth 'evolution' process, as all that is currently available is a list of stats that increase slightly if the organism 'consumes' food. This is evident then that there are very few factors that contribute to how an organism evolves over time. In my program, I am going to go for the opposite approach – focusing more on the evolutionary process instead of the graphical side. This means I can provide a more scientific software that the stakeholders can use.
- Within my project, I will be instead using a 2d visualisation of each organism displayed on a simple map, due to the fact that each organism does not need to be graphically visualised in depth, which would take away from the prime focus of the project; the ability to demonstrate evolution over an extended period of time

Limited with the amount of random chance and external factors:

- For example, within The Evolution Simulator, there is very little in the way of physical factors which affect each organism, which decreases the programs ability to demonstrate adaptation. In this program, each organism only has to eat and make it to a wall, in order to adapt. This means that there is little in the way of random chance and computational 'thinking', hence the aforementioned linearity of the program
- My program will incorporate a range of factors to calculate adaptations, such as environmental situations (hot, cold, water, no water), as well as population (breeding, disease, predators). This gives a much broader range of evolution, and opens up into the scope of AI, which will control each organism over time to find the best suited environment for evolution.

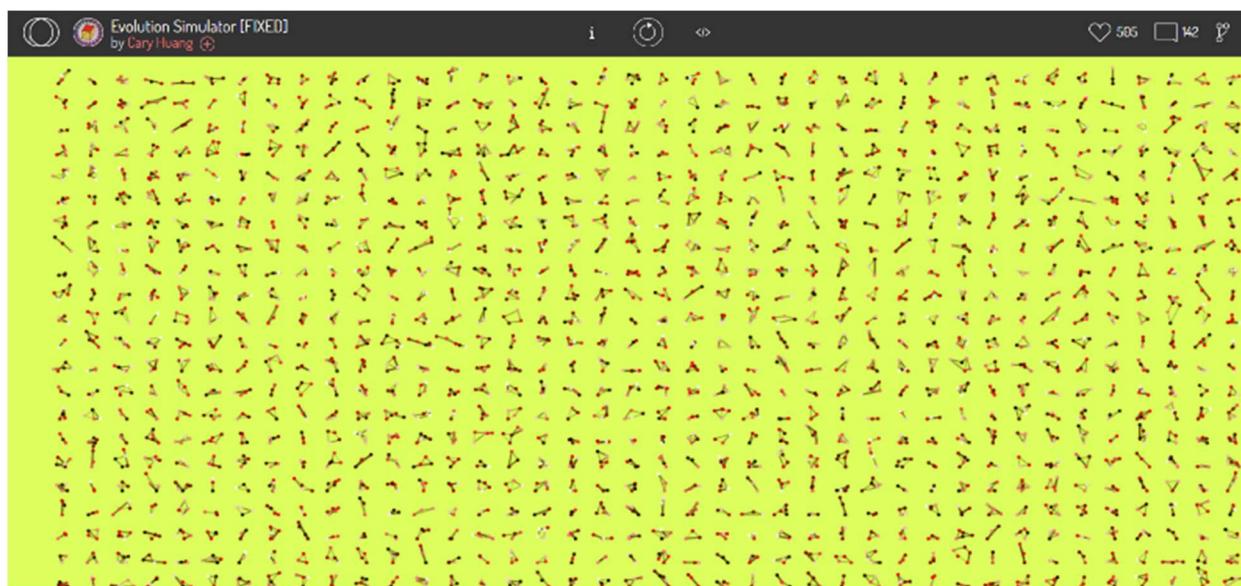
1.3.2 Evolution Simulator by Cary Huang

This example is vastly different to previous ideas, however, shows off the same concept (of evolution). This simulator creates 100 random 'joints', testing each one on their speed, with the fastest having a greater chance of survival, and the slowest having the least chance of survival. The simulator then kills 500 and reproduces (combines existing) to create another set. This process continues until there is a set of organisms which are the fastest.

Despite the fact that this simulator is different in concept from the previous ideas, there is still a few features that I would like to use from the simulator that were done well. These are listed below along with some examples of what I want to change/not use.

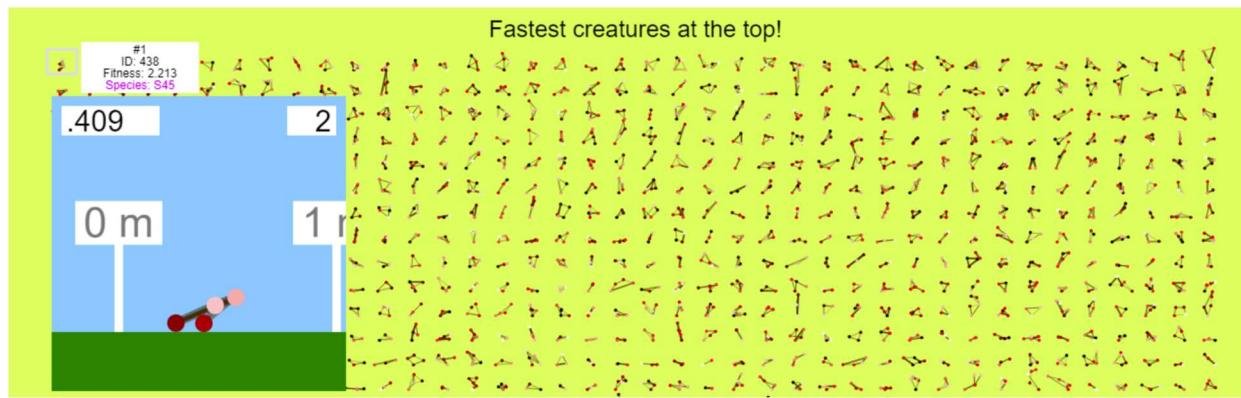
Features I would like to use from the Evolution Simulator:

- The Evolution Simulator in this example is optimised at calculating and representing the evolution process, where after breeding, the organisms that survived the previous iteration, create offspring that are potentially faster and therefore have a higher chance of survival.
 - o This example demonstrates the principle of evolution very well, displaying the organisms in order of speed (the only attribute contributing to survival). This feature is good at displaying *how* the organisms have survived, and also shows what they have evolved from overtime. In this example probability is also calculated surrounding the survivability of each organism. This example is good to base the principle around, for example how the organisms evolve (breeding) and why they evolve (speed).
- The GUI for this program is simplistic and does the job that is required to do. For example, the beginning of the program, is simplistic, with basic buttons presenting options for the creation and making of each organism. The Evolution Simulator here also runs off of a website, as opposed to my planned application design. Meaning some of the features that are here may not be available to me because of this difference.
 - o The simplistic GUI and animation of each organism is something in which I want to use for my application since the focus here is not on the GUI, instead it is on the actual processing side (which this website also does well). Using simple buttons for the opening menu as well as settings will be more than suitable for EvSim. I had no issue navigating the website while researching, further giving more reason to use a simple GUI. The website application here appears to have been built using Unity (a well-known game engine), which I believe has been used to utilise the physics side of each organism (calculate speed).



Features I would like to omit/change from the Evolution Simulator

- The Evolution Simulator here is very limited on what is calculated as a representation of evolution. Each organism is made at random, and then is tested for how fast it can go in a straight line (see figure 4 below). Despite giving a brief representation of the evolution process (fastest/most able survive), it is very limited, and does not calculate any other attributes.



- o Within my program I would like to calculate more attributes for each organism, such as strength, speed, age, preferred climate etc (see 1.4). Having multiple attributes such as the ones aforementioned means that evolution is more accurately represented and therefore can be used by the stakeholders to analyse the stages and processes of evolution itself. Furthermore, more attributes mean there is more chance for random evolution, increasing exponentially the possible variations that EvSim can end up with.
- Another feature which I would not like to use from the Evolution simulator, is the linear aspect of it. The example here is extremely limited not only with the number of attributes, but also how they are represented and how they are removed/created.
 - o In the Evolution Simulator here, each organism is generated at random, tested on speed, and then is given a probability (depending on their speed) of survival. This is where the linearity comes from, with each organism not taking in external factors, instead being given simply a probability within which they live or die. Within my program I want to give each organism the chance to evolve into a predator (small chance), to also represent other factors of death other than just attributes and percentages.

1.3.3 Evolution for Dummies, Greg Krukonis, Tracy Barr - Wiley

Due to the lack of suitable evolution simulators which are available online, I have instead chosen to include a textbook example/detailed analysis of evolution. Evolution for dummies includes key information which can help an individual understand evolution. Considering it is used to teach students about evolution academically, the same aim as EvSim, it would be ideal for ensuring that EvSim is scientifically accurate.

The example below shows an example which I will use from this textbook. EvSim will contain the ability for organisms to reproduce asexually and sexually. The implementation of this should be accurate to scientific knowledge, and as shown in the image below, asexual reproduction outlines that the offspring only contains genetic information from the singular parent. This should be repeated in EvSim.

✓ **Sexual reproduction:** *Sexual reproduction* is the system in which two individuals mate and produce offspring whose genes are a combination of some of the genes from each parent. Sexually reproducing organisms produce offspring that have only half of each parent's genes.

✓ **Asexual reproduction:** *Asexual* organisms produce offspring without mating, and these offspring contain only the single parent's genes.

Features I would like to use from Evolution for Dummies:

- As mentioned previously, I will be primarily using the knowledge contained within Evolution for Dummies to aid the development and accuracy of EvSim. The example of asexual reproduction aforementioned is one of many examples which could be used in EvSim.
 - o Another example which can be used is shown below, the effect of an organisms environment on their development. A key part of EvSim, is the map generation and effect it has on the organism. In Evolution for Dummies, it makes it clear that the physical characteristics of an organism, is the result of an interaction between its genes, and the environment itself. I aim to use this in EvSim, with the characteristics of the organism having an impact alongside the environment itself. This can be achieved through the food an organism eats for example. An organism may have a food type preference, but the environment may not have this, meaning that the organism may evolve as a result.

The effect of environment

An organism's *phenotype* (physical characteristics) is a result of the interaction between its *genotype* (genetic makeup) and the *environment*. A person with the genetic potential to be 7 feet tall won't achieve that height in the absence of proper diet, for example. A malnourished person will be stunted compared with a genetically identical individual who was well fed. The impact of the *environment* on the developing organism is called *environmental effects*, and one way that the *environment* can affect phenotype is by affecting development. (You can read more about what affects phenotype in Chapters 4 and 7.)

Details such as this I can use to ensure EvSim is scientifically accurate, and therefore more likely to achieve its goal of providing a visual means of education around evolution.

Features I would like to omit/change from Evolution for Dummies:

- There are however, certain aspects which I would not like to include from Evolution for dummies which are not required for EvSim, or not possible to implement.
 - o Evolution for Dummies aims to educate those in primary and secondary education about not only the basics, but the in depth details of evolution. The level of detail which the textbook goes into is expansive. As seen below, the book details about reproductive choice between two mammals. More specifically, the competition that occurs between species when finding a mate. Within EvSim, this level of depth will only increase the amount of work involved in implementation, and will not add any major educational gains. A main factor contributing to this is the fact that it will be hard to show that 2 organisms are competing for a mate on a large map with over 50 other organisms.



Sexual selection has two components: choice and competition. For the vast majority of cases, this means female choice and male-male competition. These are not mutually exclusive; you can see both in the same species and sometimes you even see the reverse (male choice and female-female competition). Also remember that sexual selection is found in lots of animal species from insects to mammals.

Details such as this are not relevant to EvSim, as they do not actually benefit the end experience, and requires a significant amount of work to achieve. There are many other features and examples which can be used, but the reproduction process is a prime example due to the depth.

1.4 Essential Features

Taking into account the previous research and analysis of stakeholder requests, the list of essential features is quite large. Within this section I will go through each feature individually, explaining how it will work, the use case, and why it is needed, in context of the stakeholder requirements, and linking to what I have seen so far from the 2 researched applications.

The first and most essential feature is the evolution process itself. This process is a considerably large one to consider, due to all the attributes and features that contribute into it. Within EvSim, the aim is clear; demonstrate to the user how evolution takes place over time between organisms. In order to correctly demonstrate this, the evolution process will have to be robust and justified. At the beginning of each iteration, there will be a randomly generated 2d world map. This map would then affect the attributes of each organism and their possible evolution process. For example, an organism that spawns into an ocean/water biome, will be unable to evolve land movement abilities (legs), the same way that an organism that spawns on land, will be unable to swim fast. This means that the evolution process is not entirely random, instead taking into account the surrounding environment and the attributes that this gives.

By having the environment affect the organisms individually, there are considerably more outcomes that will be created during each iteration of the program. There is also therefore *more chance for user customisation and creative freedom*. Linking in the world map with user defined attributes such as chance for natural disaster or chance of predators, allows for the end user to be able to explore every element, and therefore be able to conduct research and test hypotheses using this program, which is what the stakeholders want.

The second and just as essential feature of EvSim is the organisms themselves, and how they behave. This is also another complicated area, as each organism will be controlled by an algorithm, which strives to find the best environment for themselves. For example, linking with the past feature, an organism which was spawned in water, will want to stay in water throughout the iteration, therefore in the event that they are deprived of water, they will attempt to find their way back. The criteria controlling the organism would also affect how each organism breeds and acts as a whole. For example, an organism which has been able to develop its speed and strength most, will be more likely to survive against predators, and therefore more likely to breed. The benefits of having organisms being controlled in this way, is that their chance of survival is not dependent on chance, or probability, rather it is based on computational decisions that are explicitly based off of decisions made by the computer. Similar to the first essential feature, this means that the end user is presented with a lot more information, as opposed to random probabilities and set out factors. One factor which may benefit the stakeholders and end users more is the ability to change how the AI acts as well as the effects of the environment. This would allow them to understand the effects of both factors, and how they affect each other.

Linking both previously mentioned features back to the applications that have been researched in 1.3, the main difference and benefit of having AI controlled organisms, and a randomly generated map, is that

there is a lot more random chance, and more options for outcomes. This is what the existing applications failed to accurately represent and is why there is no currently applicable program for the stakeholders to use. Further essential features will be outlined below.

Individual elements of each organism, and their attributes from evolution, is once again one of the most essential features for this program, as it is what is evolving. Each attribute will be calculated per organism per cycle (year^{*Subject to change}), and will be slow to change per cycle, due to the amount of time it takes to evolve. This is beneficial for the program for multiple reasons. Firstly, having each organism evolve slowly provides more accurate information per cycle, and also allows for the iteration process to not be too short. This also benefits the running of the program itself, due to the number of calculations that have to be calculated in a short space of time. Attributes that are going to be calculated are listed below, with a brief explanation as to why.

- Initial and final form – Due to the project not being designed in 3d, we will be using shapes to represent each organism's development stage. This is important for visualising the stage of evolution that each organism is in, while also showing which evolution path the organism has taken.
- Retention rates – Retention rates evaluate the amount of water/oxygen an organism can keep at a given amount of time. For example, an organism with high amounts of oxygen retention, is developed for the ocean territory, whereas one which has oxygen retention rates, will be developed for the desert biomes.
- Age – Age is one of the most important ones to consider, due to the fact that it represents the survival of an organism, and how their attributes have allowed them to survive. The more developed an organism is in an environment, the longer the organism will be able to live and the more chance the organism has of breeding.
- Main food source – this is more focused for the user (requested by stakeholder), and will categorise what the organism eats, this will be calculated depending on the environment, for example an organism in a jungle territory will be more likely to be an herbivore.
- Health – This is once again in relation to all other factors, for example an organism that has a lot of food and water available, will have better health and receive a buff to the other factors such as speed and awareness.
- Speed – This is in the event that predators are evolved too, as an organism will need to either be fast (as prey), or fast (as a predator).
- Awareness – Similar to the speed attribute, it is more focused towards the predators and prey side (can be disabled by user), for example prey will have to be aware of predators as much as predators will have to be aware of prey.
- Size – This increases depending on other attributes such as health and age, as an organism evolves overtime, it may grow depending on its environment and it may also not.
- Attack – This is once again focused on the predator side specifically, as the more developed a predator is at attack, the easier it is for them to find prey and eat.
- Defence – As a posed to attack, this is focused on the prey side, with the organism having a higher chance of survivability and sustained health with a higher defence. Organisms that survive attacks will develop their defence attribute a lot faster compared to an organism that is never attacked.
- Habitat preference – This attribute is entirely dependant on every attribute combined, with the preferred habitat being that in which the organism has the highest health and highest chance of survival.
- Food preference – Food preference links in heavily with main food source, however most organisms will be able to eat as they please.

- Temperature preference – This is more in relation to the biome in which an organism spawns in, for example one which spawns in an arctic biome, will develop preference for cold weather, as opposed to one which spawns in the desert.
- Survivability (current biome) – This uses every attribute, and takes into account other organisms near by, to give a survivability estimate for each organism. This will be higher for an organism which has the most developed attributes (most evolved).

All of these factors mentioned above are ones which have been requested by the stakeholders, however, due to time limitations and programming complexity, not all attributes will be able to be calculated, meaning the most significant ones will be focused on first. The end aim is to have all available, however this may not be possible.

Other less essential features, but still required for the end product, are terrain specific evolutionary contributors. This once again links heavily into the first 2 features, the map, and the AI control. Firstly, the randomly generated map should have areas with different contributing factors such as temperature and visibility, much like real life. This would add another element to both the simulation and the evolution process, as organisms are not only adapting based on their own factors, but external influences. Other factors such as seasons can also be introduced to influence an organisms choice of terrain and place to stay. If an organism survives in one biome for an extended period of time, they may also receive alternate benefits once in those terrains. For example, if an organisms ancestors have lived in a terrain for generations, then the child organism will have extraordinary adaptations to that environment, such as a greater temperature tolerance.

- These features are essential for representing a realistic evolutionary process (the aim), therefore beneficial for the stakeholders. In the case that seasons are not needed/wanted, there will be an option to disable them, meaning that the stakeholders have a choice of what they want to test, giving them more control over the program. This is also one of the features requested by the stakeholders (the ability to change factors in the simulation).

One of the most important visual features would be the graphical interface itself. This has to be more functional than complex, as it is not the main focus of the project. This means that the end product will only have a basic GUI, that provides more functionality to the user. For example, as requested by the stakeholders, the ability to save variants, load, pause, speed up and slow down the pace of the simulator is important. This means that the end product will have a starting menu, asking the user whether they want to load an older variant, or start a new variant. Despite this requiring more resources, it is necessary for the end user and their need for this function. Furthermore, the GUI will have to include the ability to change probabilities locally/globally, in order to allow the stakeholder to carry out observations. For example, the user should be able to view the factors globally through a side menu option, and view the factors for individual organisms by clicking on them and then using the side menu. This will only be possible through correct representation of the map and the organisms. I plan on representing the map in a 2d space, where the user will be able to navigate around (either using the mouse or keyboard), as well as zoom in (to an extent), and zoom out. This allows the user/stakeholder to more accurately observe/view elements of the simulation individually. Combining all of these so that the user can pause, zoom in, save, and then analyse attributes will be especially important for the stakeholders to carry out their observations and test hypotheses.

1.5 Limitations

Limitations within a project as large as this one are bound to be present, for example time constraints may pose an issue with the amount of attributes that have to be calculated. Furthermore, my current knowledge on the processing behind a GUI is lacking. Limitations like these will be listed out and explained below.

One of the most prominent limitations of a project like EvSim, is the sheer amount of attributes and probabilities that need to be calculated consistently in such a small amount of time. For an evolution simulator to be as detailed and accurate as possible, there are a lot of factors that need to be considered. These factors also have to have an impact on each other, for example the external environmental factors influencing the organisms traits. This is where time constraints become an issue, with the amount of factors that have to be calculated, some may have to be omitted from the end planning, in order to still deliver a finished product, even if it is not fully fleshed out.

Furthermore, each individual factor has to be researched, and planned out as a tree as to how evolution would work, in order to make it accurate. This would also affect the end result regarding how many attributes there are. Time constraints during the research for attributes, as well as the design stage of how the attributes will be calculated in, will take up a large proportion of time. This is also why I have chosen to go with a basic design for GUI, in order to save time during the design.

Another limiting factor is the GUI itself. Expanding on previous comments regarding graphical interface, my limited knowledge on GUI design and creation is another reason to not create a 3d scape, such as the one seen in 1.3.1. If I was to design a 3d scape, time would be taken away from the attributes and therefore the goal of the project, meaning the project would not achieve what it set out to do.

1.6 Hardware and software

1.6.1 Hardware Requirements

Hardware requirements are needed in order to consider what the stakeholder will need to test the project at the end, and also ensures that the project is not too resource intensive, as this is expensive and not globally available to most stakeholders.

For my project, due to its heavy focus on processing, it requires a processor with at least 1.5Ghz clock speed, and a minimum of 2 cores. These factors are the most important due to the amount of calculations that will be processed per second, and also determines the speed at which the program can be ran. The better the system processor, the faster the simulation can be ran and therefore the quicker the stakeholder can get to results/testing hypotheses. A minimum of 2GB RAM is also required due to the amount of libraries that have to be imported into the program for factors like random map presentation. RAM is also required to handle processing of the program, and the updating of the GUI. Overall, the program shouldn't take up more than 200mb of storage, as there are no large files such as images or models.

As limited by GUI programming, I will require the user to have a recommended 16:9 aspect ratio monitor, with the preferred being 1920 x 1080. This is because the simulation will be run through a window set to the 16:9 ration (by default 1920 x 1080). Anything outside of this range is likely to not fit on the screen, and therefore can result in a decreased user experience, or missed information. As this project is designed to help students in school, a larger monitor will be recommended, especially due to the vast amount of organisms and factors that can be seen through the screen, for example biome and forms of each organism. In order to aid with this, the program will include a zoom function, as well as the ability to move around the map.

This therefore requires the user to have a keyboard and mouse in order to operate the program. The keyboard will be used to move around the map, as well as zooming in/out. A mouse will be used to select individual organisms, whereby attributes can be viewed. A mouse will also be required to change the attributes (as requested by the stakeholder).

1.6.2 Software Requirements

Software side, I will be programming EvSim in Java, as it is a familiar language to me, as well as it being catered to processing intensive programs, and the large number of external libraries that are also available. This means that the user will have to have an operating system capable of running Java, for example Windows 11, or MacOS 14. Despite these systems being the latest and most popular versions, it is not restricted to these operating systems, as EvSim can be run on the majority of Operating Systems all the way back to Windows 7. External libraries may be used within the program, one of the main ones being the Java Development Kit 17, as it includes an abundance of libraries which will be used within the program. Due to the side of jdk-17, the user will have to download this themselves from a link provided, however any other smaller libraries used for the development will be implemented alongside the main package.

1.7 Success Criteria

The main aim of EvSim, is to provide students and teachers will a sufficient and simple program which allows evolution to be visualised. Evolution is a hard concept to get around, and therefore the program has to be able to visualise it in a simple and understandable way. The app will be used by both students and teachers, and therefore it will need to have an understandable UI, with potentially instructions imbedded to help with the understanding.

In order to consider the project a success, EvSim will have to have the following requirements met:

- 1) Allows the user to change and set attributes during the creation of the random map and organisms
 - I. Allowing the user to change and set each attribute means that the creation process does not have to be random and therefore makes the overall layout and design easier, both front end and back end. On the front end, the user can decide to change the attributes themselves (allow for specific testing), or go with the base values, which will be a realistic and natural scope (with attributes such like real life). On the back end, it will be easier to define the attributes as an input from the user, rather than multiple random calculations, potentially leading to highly unrealistic scenarios. Having a random chance set, would be a luxury in this case rather than a necessity.
- 2) It must be able to load and save currently processing formats, as well as the option to pause and change speed of the simulation.
 - I. For the program to be able to load and save the information currently being stored means that the software is much less vulnerable on the front end. For example, if the program was to stop for any reason outside of the users control, then the user would not lose their progress and render the time spent useless. Furthermore, being able to switch between different simulations means that it is far easier to compare and therefore test hypotheses, which is what the end user (stakeholders) want to do.
- 3) It must 7 of the organism attributes mentioned in 1.4 Essential Features.
 - I. Displaying the necessary attributes (of 7) is important due to it being the main focus of the program. Without the display of attributes, the program is simply different colours moving and changing on a 2d plane. Without the ability to display the attributes of each organism, the stakeholders cannot carry out their tests and therefore have no use for the program, meaning the solution to the problem mentioned in 1.0 is not fulfilled. Evidently, displaying attributes is required for this software to be marked useful.
- 4) It must be able to create and display a world which has fluid biomes similar to what could be found on Earth. For example grass biomes should contain forests, and oceans should be surrounded by a beach. The world should also contain the different organisms.

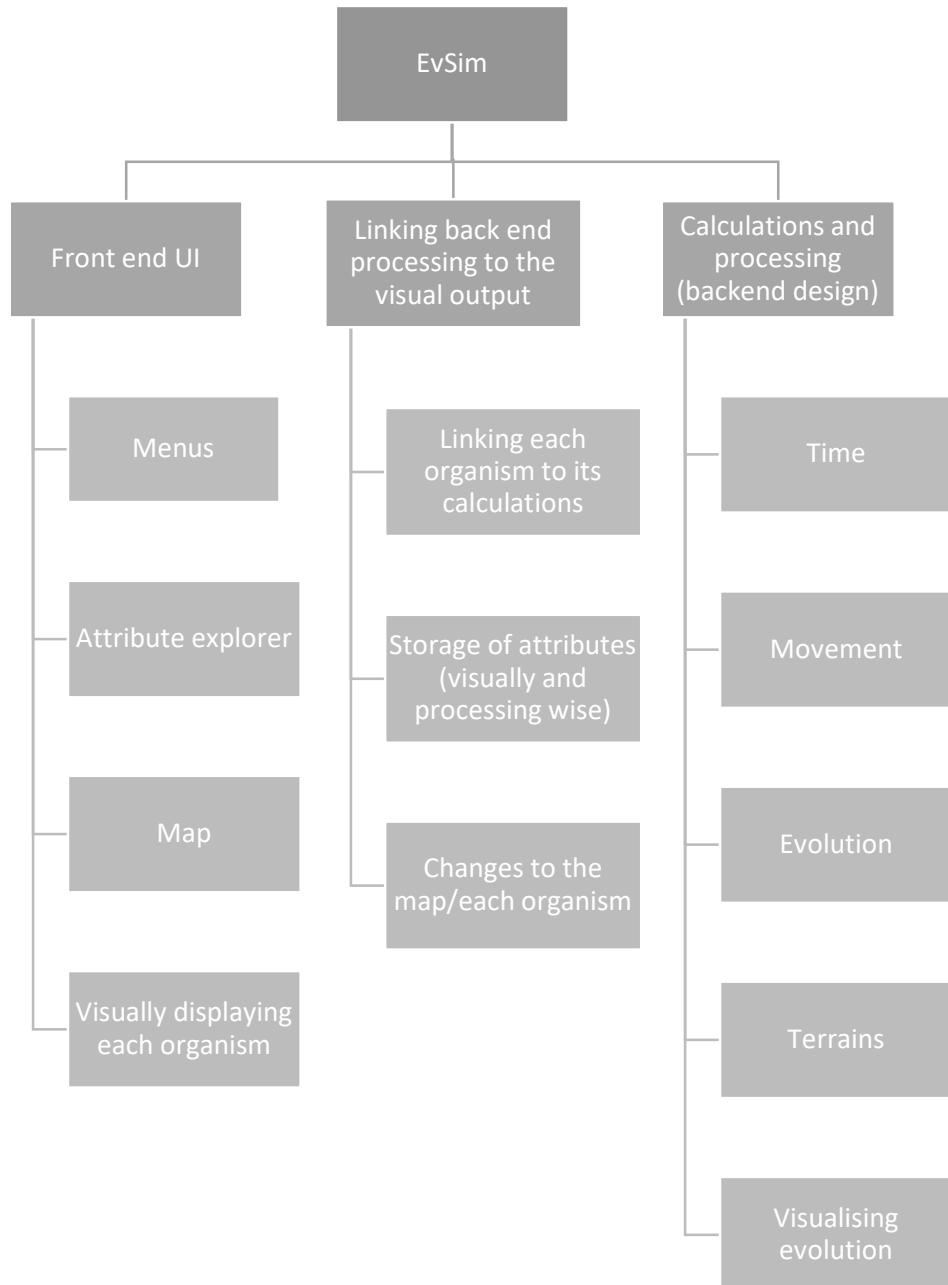
- I. By displaying a “world”, whether that be in detail or in visible set areas, it is necessary for the program to visually make sense to a user. For example, the organisms are going to develop based on which environment they are located in, and therefore the user also need to know which environment they are in, to fully understand the reasoning behind the evolutions that have taken place. The easiest way to demonstrate this, is through a visual map. Furthermore, by having a visual map, the program has something to display, rather than potentially a line of code (at the bare minimum), or a list of organisms (which would look like the program in 1.3.2).
- 5) It must be able to demonstrate evolution both physically and visually. Visually through the organisms changing shape, and physically from their factors developing depending on the biome they reside.
- I. With this factor being perhaps the most self explanatory, by being an evolution simulator, it would only make sense to have it demonstrate not only an accurate version of evolution, but a simpler version. If the program was to have an in depth and complicated flow of evolution, the end user may not be able to use and understand the program as well as they could without the complicating factors, and therefore it is chosen to go with a simpler and more linear (laid out) approach.
- 6) It must be able to sufficiently calculate and store the change of attributes, so that the program can display the change visually to the user.
- I. Another needed factor, with this one being mainly the backend of the program. For the program to achieve this criteria, the program has to run at a sufficient pace (optimised), and it also has to allow for the previous point to also be met. All calculations and variables that are used will need to be efficiently calculated and processed in the least amount of time possible in order to provide the end user with a useable and fast piece of software. If the program runs slow then it will be too inefficient for the end user to consider using, and if the program is too inaccurate, then the end user cannot use it to make evaluations and tests (which is what the stakeholders want from the program)
- 7) It must be able to operate sufficiently with changing parameters, for example an organism moving between biomes should be taken into account during the processing, as an organism can move at the same time as its attributes are being updated.
- I. Taking into account the previous statement, the programs calculations will also have to take into account changes in parameters and factors, while the processing is happening, for example an organism moving from one biome to another. This links in with the previous point whereby processing and validation must take place efficiently to provide a smooth and accurate operation.
- 8) It must be able to take inputs from the user during the execution of the simulation.
- I. The program must account for the user’s desire to change factors during the simulation, such as terrain specific and organism specific factors.
- 9) It must be able to demonstrate evolution over ‘generations’ of organisms, with elements only being available after an extended period of time.
- I. This is more specific to evolution. Factors must update slowly over time and generations. This means that any new organisms must take attributes from their parents respectively, and also have adaptations only available after longer periods of time. Ideally, the program should be able to execute infinitely, with evolution continuing throughout.

The criteria laid out here is chosen to meet either the stakeholders demand, or to match the personal coding ability, with each factor being justified.

2.0 Design

2.1 Problem Decomposition

To simplify the design of EvSim, I have divided the program into 3 sectors, as demonstrated below:



2.1.1 Front End UI

The front end UI is one of the primary features of EvSim, as it is what the user will see. Breaking this down further, the first section of Front End UI, is the menus. The menus will have to be developed first, so that the simulation can be loaded as intended. The menus will also be required to be implemented first so data entered by the user is present, for example the generation of the map. The attribute explorer is an element of the UI which can be left towards the end, the main reason for this being the fact that the attributes are not yet decided. However, the map and visual display of the organisms can be completed at the start. One of the first tasks of development is to implement the map. The map generation is also an undecided factor, meaning the user data which is entered is not yet known. The

display of the organisms should also be done relatively early into development so that evolution and movement can be tested.

2.1.2 Linking Data

I have decided to make a separate subtopic for the front end to back-end link, due to the amount of data having to be stored and used at the same time within this project. For example, if we have a simulation of 100 organisms, each one having 7 attributes, and 5 biomes are on the map. There are going to be at least 3500 calculations happening (before computational methods mentioned in 1.1 simplifies this). Within this subtopic, I have decided to define changes to each organism, storage of attributes, and the linking of an organism to its calculations as separate areas. Despite all of these being different areas of the program to each other, they all come under the umbrella of data and storage moving between the front end and back end. Back end storage will hold information for all of the organisms in separate file, holding for example the organisms attributes as well as the map itself. Storing all this information in a file means that information can be read and transferred across classes easily. It also creates a physical store of information which can be read in the future to load an old simulation, a factor of the 1.7 Success Criteria.

Evidently, due to EvSim being a data heavy program, there will have to be a sufficient method of data storage and access to allow for a smoother execution of the end program. The link between front end and back end will also have to be fast, as time cannot be wasted on execution reading data from the file.

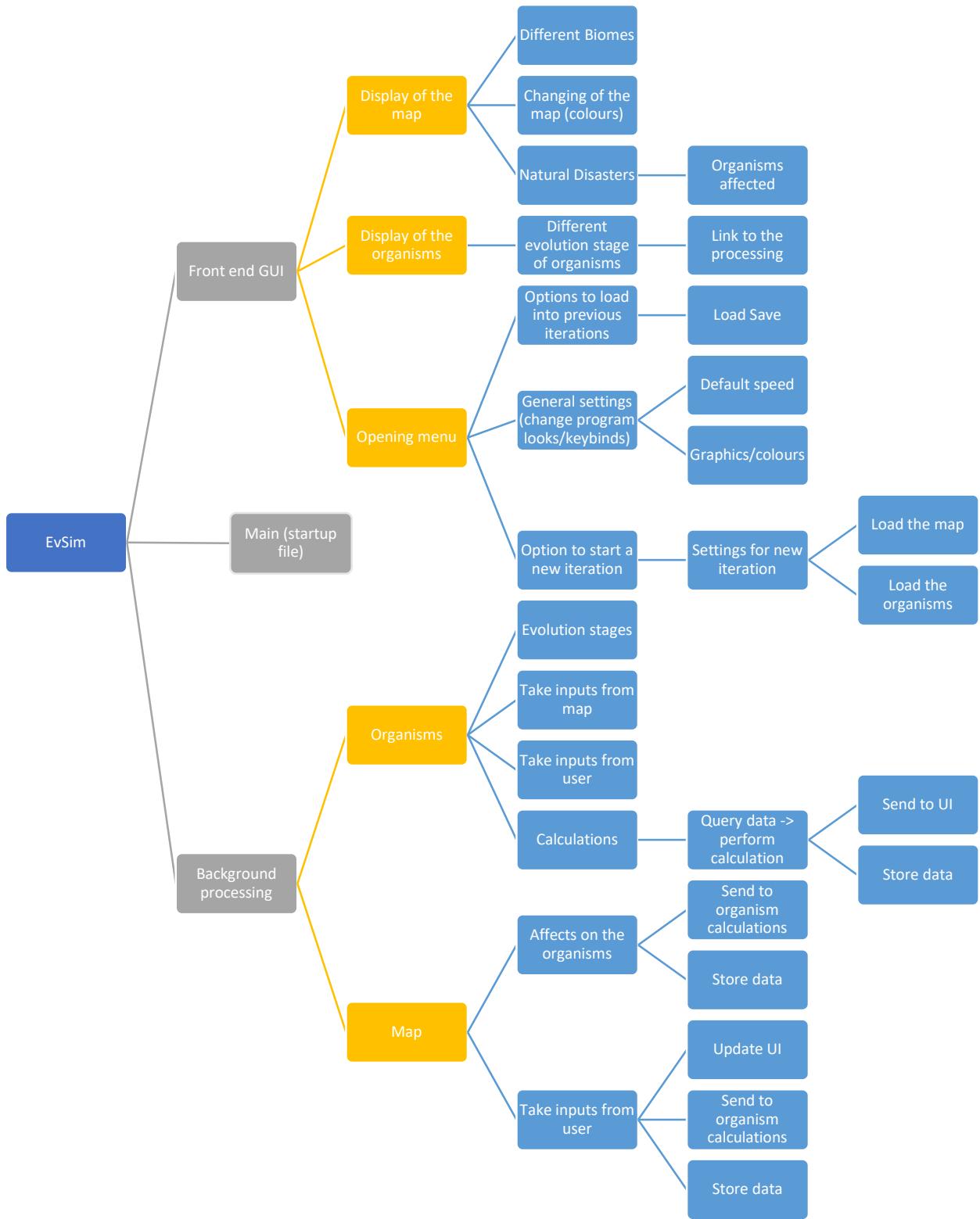
2.1.3 Calculations and processing

With the calculations and the background processing being the backbone of EvSim, it was very clear early on that this stage would have to be separate from other sections of development. The system must take inputs from the user, whether that be changes to the environment, or changes to how fast the system should run. All these factors must be considered when looking at the background processing. A few of the subtopics mentioned underneath the processing category, such as individual organisms and biomes are of course dependant on the user inputs, this means that changes must be constantly checked for, and then adequately changed in accordance with what the user wants. Time is an example of this – the user should be able to change the speed EvSim processes, so that evolution takes place over a long or short period of time. The simulation should also be able to be paused. Considering these factors, it is evident that the best way to proceed in the development stage is to focus on each of these factors separately, in order to not cross interests (during development) and repeat code.

Breaking down a program as large as this one is beneficial to not only visualise the programs development better, but to also aid in the understanding of how the program will piece together. Having the basic structure laid out first allows for further expansion and detail to entail. A hierarchical structure like the one demonstrated here is perfect in this situation in order to begin the Decomposition and Selection process mentioned in 1.1.2. Planning for the development stage is the key to not only an efficient program, but one that is simple to read and write.

2.2 Structure of the solution

Below is a horizontal hierarchy chart of how my program will be decomposed:



Following on from the root of this project (EvSim), I have broken down the project into the background processing and the front-end GUI, as the linking between the two cannot be separated easily. For decomposition, it provided a visual representation, however in the structure, it is not required to be separate. EvSim and the main file will act as the startup file for my program, containing the root sources (libraries) and necessary files (code).

Starting with *Front end GUI*, this is where the smaller portion of success criteria are met. For example, within the *Opening menu*, we can find the area within which users can load up a previous iteration of

the program, as well as change the default settings such as speed. The ability to load into a previous game was not only one of the stakeholders' major requests, but also a factor of the success criteria, so we are meeting those points by having them laid out in the main opening menu.

Furthermore, the *Front end GUI* also contains process within which the information is displayed for users, for example the map itself. Within the *Display of the map* section, the project will update the visual output for the user, in three bases: *the biomes*, *the changes of the map*, and the *natural disasters*. Within the biomes, the main map is set out (whether that be from a save file, or a new file (random generation)), and therefore the individual attributes (per biome) are stored (for processing later). For example, an organism that is developed for land, once it enters a water biome, will no longer have a high survivability attribute (mentioned in essential features). The biome stage is important, as it sets out the stages of evolution for organisms in the future. The points mentioned here are similar for the changing of the map and natural disasters.

The final area of the *front-end GUI* is the *Display of the organisms*. Similarly, to the display of the map, where the organism is located on the map, correlates to its biome, and therefore what values need to be calculated for its evolution. It is important to accurately link the location of the organism to what biome it is, not only for visual purposes, but to make sure that the organisms are developing sufficiently. This is why a subgroup within the display of organisms is the link to the processing.

The *Background processing* area of EvSim is set out between Organisms and the Map. Similar to the GUI, this is where the two will need to link. The front end in this situation is responsible for visually showing the user what is happening in the background processing side. Expanding into the background processing, specifically the organisms, the structure is defined from mainly taking in the values from the map/biome and performing calculations on that. Within the calculations subgroup, the data is queried (selection), and then stored (*store data*), and sent to the front end, *Send to UI*.

Similar to organisms, the map is also one which have been divided into affects and inputs. There are less inputs from the map, with those only being when the user changes attributes throughout the running of the program (mentioned in 1.4 Essential features). Once the user changes the attributes while the program is running, the *Map* subtopic is responsible for updating the front end (*update UI*), sending inputs to the calculations (*sending to organism calculations*), and *storing the data*. The individual effect on organisms is similar, as it takes the user inputs per organism, and sends the information to *Organisms*, to be calculated and stored (*Store data*).

In a simple form, the background processing is responsible for performing calculations and taking inputs from the user, whereas the front-end GUI is responsible for updating the visual interface depending on the calculations and inputs.

2.3 Algorithm Design

There are going to be many difficult elements to the development of EvSim, from the generation of the map and its biomes to the movement and development of individual organisms. This design section is going to cover the majority of the algorithm (design) for these parts. It is likely that these designs will change throughout the development of EvSim, as new factors and problems are introduced. The first part of the algorithm design section is the map generator.

2.3.1 Map Generation

```
1. // MapGenerator.java
2. // This file opens a new game window, generates, and displays a random display of 2 biomes;
Grass, and Water
3.
4. public class MapGenerator
5.     private int WIDTH = 1920 // the width of the map/game panel
6.     private int HEIGHT = 1080 // the height of the map/game panel
7.     private int cellSize = 5 // the size of each map tile
8.     private int numberOfRows = HEIGHT / cellSize // each row is the size of a cell
```

```

9.     private int numberOfRows = WIDTH / cellSize // each column is the size of a cell
10.    public int[][] map //the map grid itself
11.
12.    private procedure generateWorldMap() // defines what biome each map tile should be (as an
integer)
13.        int grass = 0, water = 1, row, col, terrainType, chance
14.
15.        for row = 0 to numberOfRows
16.            for col = 0 to numberOfRows
17.                terrainType = grass //sets the default biome to grass
18.
19.                chance = random(0, 100) // random number between 0 and 100
20.
21.                if chance < 50 then
22.                    terrainType = water // 50% chance of biome being
water
23.
24.
25.                map[row][col] = terrainType
26.
27.            next col
28.        endprocedure
29.
30.    public procedure MapGenerator() // paints the map to the game/app window
31.        Panel gamePanel = new Panel(HEIGHT, WIDTH) // opens a new window the same size
and width as the map
32.        map = int[numberOfRows][numberOfCols] // initialises the map integer array
33.        generateWorldMap() // generates the map/random biomes
34.
35.        int colourToPaint
36.
37.        for int row = 0 to numberOfRows
38.            for int col = 0 to numberOfRows
39.                if map[row][col] === 0 then // if the biome is grass (0),
the colourToPaint is green, else it is blue
40.                    colourToPaint = RGB(0, 255, 0)
41.                else colourToPaint = RGB(0, 128, 255)
42.
43.                gamePanel.paint(col * cellSize, row * cellSize, cellSize,
cellSize, colourToPaint) // paints an object -> .paint(x, y, width, height, colour)
44.            next col
45.        next row
46.    endprocedure
47. endclass

```

pseudocode

This pseudocode is the base level idea for the map design. This code is designed to produce a basic visual map, with the two biomes being Grass and Water. Despite this being primitive, it allows for easy expansion, specifically for biomes, with the ability to add in a new biome, as well as the ability to increase the chance of it decreasing or increasing. The code has two procedures, one to generate the map, and the other to paint that generated map. The procedure MapGenerator, calls generateWorldMap, and then paints what has been generated.

2.3.2 Organism generation and movement

This section has quite a lot to be covered, beginning with the getTerrain() method.

2.3.2.1 getTerrain()

```

1. public function getTerrain(x, byval; y, byval)
2.     return map[x][y]
3. endfunction

```

pseudocode

This function can be placed in the MapGenerator.java section, or in another file (will need to still fetch from MapGenerator.java). This section is designed to return the terrain type where the organism is currently at. The function takes in the x and y values, and returns the integer (terrainType) at the same

coordinates. Returning the integer at which the organism is currently at is vital in order to link the attributes of the biome to the organisms that are currently in it.

2.3.2.2 OrganismGenerator.java

The method for generating organisms here will be relatively similar to how the map is generated. This time however there will only be a user defined number of organisms, at random locations across the map. For each organism that is generated we will be also creating the base layer for the stats to be created too.

```

42.                                         "preferredFood": null // This
will change depending on what biome the organism starts in
43.                                         },
44.                                         "preferredTerrain": null,
45.                                         "isDead": false
46.                                     }
47.                                 )// The object created in Organisms.json
48.                             next countOrganisms
49.                         endprocedure
50. endclass

```

pseudocode

This file is essential for both the ability for EvSim to run, but also for current iterations to be saved and loaded. This is primarily due to the JSON object array which is created. Having the current state of all elements saved to a physical file, means that it can be accessed across files and classes. This also means that when EvSim starts, the previous state of organisms can be loaded again, if saved previously. This meets the second part of the success criteria.

Using the information in the JSON file, and using the information generated within the map generation, everything can be painted from one class, allowing for consistent updates to everything as the time shifts.

2.3.2.3 Organism Movement

This is a relatively small section of the main file, where the organism movement is calculated. This demonstrates the usability of the JSON file as a method of storage.

```

1. // updateOrganism()
2. // This section will update the x and y coordinate for the specified organism within
Organisms.json
3. // This means every update can be repainted accordingly.
4.
5. public procedure updateOrganism(newX: byVal, newY: byVal)
6.     int currentOrganismId = this.Id // the current organisms Id
7.     organismFile = writeFile("Organisms.json")
8.     for int currentCheck = 0 to organismFile.length // to the amount of keys (objects) in
Organisms.json). One object is represented as one organism.
9.         if organismFile.organismId === currentOrganismId then // Making sure the
organism to update is the same one as the one being processed
10.            organismFile[currentCheck].x-coordinate = newX // updating the x-
coordinate
11.            organismFile[currentCheck].y-coordinate = newY // updating the y-
coordinate
12.        endif
13.    next currentCheck
14. endprocedure

```

pseudocode

This is an example of how JSON is useful for data storage within EvSim. Each organism can be linked to an ID, and therefore all the information that is needed can be found, updated or read accordingly. This procedure here updates the current x and y coordinate whenever the organism moves, so that it can be repainted accordingly.

2.3.3 Organism attributes

This is the main section of the program, as this is where the processing takes place primarily. Below is the full class design for this section.

```

1. // Organisms.java
2. // This is the main class where all of the calculations and linking take place.
3. // Contains two main procedures, the AI control (making a decision based off of attributes),
and the stats.
4. // The final version of this in development may contain a more in depth system for stats and
AI, this is the principle design
5.
6. public class Organism
7.     private organismFile = openWrite("Organisms.json")

```

```

8.     private int Id // The current organismId
9.
10.    public procedure Organisms()
11.        // The variables that are going to be used
12.        private int preferredTerrain
13.        public int currentTerrain
14.        public int currentX
15.        public int currentY
16.        private array[] newXY
17.
18.        while gameRunning === true // To keep it updating unless the game is paused or
in the main menu
19.            for int organism = 0 to organismFile.length
20.                // setting the current attributes for the organism
21.                Id = organismFile[organism].organismId
22.                currentTerrain =
organismFile[organism].currentTerrain
23.                currentX = organismFile[organism].x-coordinate
24.                currentY = organismFile[organism].y-coordinate
25.
26.                if organismFile[organism].preferredTerrain != null
then // if the organism's preferredTerrain isn't null (meaning it is not a new organism)
27.                    this.preferredTerrain =
organismFile[organism].preferredTerrain
28.                else // The organism is new
29.                    this.preferredTerrain = currentTerrain
//sets the default preferredTerrain to the organisms current terrain
30.                endif
31.
32.                updateStats(Id)
33.
34.                this.newXY = makeDecision(this.preferredTerrain,
currentTerrain) // calls to make the decision whether to move or not
35.                this.currentX = newXY[0]
36.                this.currentY = newXY[1] // updates for the new X
and Y coordinates
37.
38.                updateOrganismFile(Id)
39.
40.            next organism
41.        endwhile
42.    endprocedure
43.
44.    private function makeDecision(preferredTerrain: byVal, currentTerrain: byVal) //
Simplified decision-making logic, in the final version, this will be further in depth
45.        int newX, newY
46.        array newXY
47.        if currentTerrain != preferredTerrain then
48.            float randomScore = random(0, 1) // random float between 0 and 1
49.            if (randomScore > 0.5) {
50.                // Make a larger move to a new location
51.                // Can only move 4 tiles in one direction maximum
52.                newX = this.currentX + random(-20, 20)
53.                newY = this.currentY + random(-20, 20)
54.
55.                this.currentTerrain = getTerrain(newX, newY) // Uses the
getTerrain function
56.
57.                this.currentX = newX
58.                this.currentY = newY
59.                newXY = newXY[newX, newY]
60.            else
61.                // Make a smaller move to move to a new location
62.                // Can only move 2 tiles in one direction maximum
63.                newX = this.currentX + random(-10, 10)
64.                newY = this.currentY + random(-10, 10)
65.
66.                this.currentTerrain = getTerrain(newX, newY) // Uses the
getTerrain function
67.
68.                this.currentX = newX

```

```

69.                     this.currentY = newY
70.                     newXY = newXY[newX, newY]
71.                 endif
72.             else
73.                 // Smaller move to stay within the same area
74.                 // Can only move 1 tile in one direction maximum
75.                 newX = this.currentX + random(-5, 5)
76.                 newY = this.currentY + random(-5, 5)
77.
78.                 this.currentTerrain = getTerrain(newX, newY) // Uses the getTerrain
function
79.
80.                     this.currentX = newX
81.                     this.currentY = newY
82.                     newXY = newXY[newX, newY]
83.                 endif
84.             return newXY
85.         endfunction
86.
87.     private procedure updateOrganismFile(int Id: ByVal) // updates the file for the new
location and new terrain
88.         for int organismIdCheck = 0 to organismFile.length
89.             if organismFile[organismIdCheck].Id = Id then
90.                 organismFile[organismIdCheck].currentX = this.currentX
91.                 organismFile[organismIdCheck].currentY = this.currentY
92.                 organismFile[organismIdCheck].currentTerrain =
this.currentTerrain
93.             endif
94.         next organismIdCheck
95.     endprocedure
96.
97.     private procedure updateStats(int Id: ByVal)
98.         switch organismFile[Id].currentTerrain
99.             case 0: terrainAttributes = new TerrainAttributes.grass // Returns the
grass class from TerrainAttributes.java
100.            case 1: terrainAttributes = new TerrainAttributes.water // Returns the
water class from TerrainAttributes.java
101.            case 2: terrainAttributes = new TerrainAttributes.mountains
102.            case 3: terrainAttributes = new TerrainAttributes.desert
103.            case 4: terrainAttributes = new TerrainAttributes.snow
104.            case 5: terrainAttributes = new TerrainAttributes.forest
105.            case 6: terrainAttributes = new TerrainAttributes.deepWater
106.            default: terrainAttributes = new TerrainAttributes.baseAttributes
107.        endswitch
108.
109.        if organismFile[Id].stats.age == 0 then
110.            organismFile[Id].stats.preferredFood =
terrainAttributes.MainFoodSource // if the organism is new, then the null attributes are set
111.            organismFile[Id].stats.preferredTemperature =
terrainAttributes.temperature
112.        endif
113.
114.        if organismFile[Id].stats.health < 100 then
115.            organismFile[Id].health = organismFile[Id].stats.health +
terrainAttributes.healthGain // if the organism is not on 100% health, then their health will
slowly increase
116.        endif
117.
118.        if organismFile[Id].stats.age MOD 5 == 0 then
119.            organismFile[Id].stats.speed = organismFile[Id].stats.speed +
terrainAttributes.speedGain // every 5 years the speed of the organism increases
120.        endif
121.
122.        switch terrainAttributes.temperature // If the temperature of the current biome
is different to the preferred temperature, then the health and the preferred temperature change
123.            case > organismFile[Id].stats.preferredTemperature + 5:
124.                organismFile[Id].stats.health = organismFile[Id].health -
0.3
125.                organismFile[Id].stats.preferredTemperature++
126.            case < organismFile[Id].stats.preferredTemperature - 5:

```

```

127.                                     organismFile[Id].stats.health = organismFile[Id].health -
0.3                                         organismFile[Id].stats.preferredTemperature--
129.                                         case > organismFile[Id].stats.preferredTemperature + 10:
130.                                             organismFile[Id].stats.health = organismFile[Id].health -
0.75
131.                                         organismFile[Id].stats.preferredTemperature =
organismFile[Id].preferredTemperature + 2
132.                                         case < organismFile[Id].stats.preferredTemperature - 10:
133.                                             organismFile[Id].stats.health = organismFile[Id].health -
0.75
134.                                         organismFile[Id].stats.preferredTemperature =
organismFile[Id].preferredTemperature - 2
135.                                         default:
136.                                             organismFile[Id].stats.health =
organismFile[Id].stats.health - random(-0.1, 0.1)
137.                                             organismFile[Id].stats.preferredTemperature =
organismFile[Id].preferredTemperature - random(-1, 1)
138.                                         endswitch
139.
140.             if terrainAttributes.MainFoodSource != organismFile[Id].preferredFood then
141.                 organismFile[Id].health = organismFile[Id].health - random(0, 5) // different food type means the organism health decreases
142.                 if random(0, 100) < 5 then
143.                     organismFile[Id].preferredFood =
terrainAttributes.MainFoodSource // small chance for preferred food source to change alongside the biome
144.                 endif
145.             endif
146.
147.             switch terrainAttributes.foodAvailability // based on the amount of food that is available to the organism, their retention and health increases/decreases
148.                 case < 20:
149.                     organismFile[Id].stats.health =
organismFile[Id].stats.health - random(2, 10)
150.                     organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(2, 10)
151.                     case < 40:
152.                         organismFile[Id].stats.health =
organismFile[Id].stats.health - random(1, 6)
153.                         organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(2, 10)
154.                     case < 60:
155.                         organismFile[Id].stats.health =
organismFile[Id].stats.health - random(0, 3)
156.                         organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(0, 3)
157.                     case < 80:
158.                         organismFile[Id].stats.health =
organismFile[Id].stats.health - random(-2, 2)
159.                         organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(-2, 2)
160.                 default:
161.                     organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate - random(0, 5) // Food is surplus, retention rate decreases
162.             endswitch
163.
164.             switch terrainAttributes.waterAvailability // based on the amount of water that is available to the organism, their retention and health increases/decreases
165.                 case < 20:
166.                     organismFile[Id].stats.health =
organismFile[Id].stats.health - random(2, 10)
167.                     organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(2, 10)
168.                     case < 40:
169.                         organismFile[Id].stats.health =
organismFile[Id].stats.health - random(1, 6)
170.                         organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(2, 10)
171.                     case < 60:

```

```

172.                                     organismFile[Id].stats.health =
organismFile[Id].stats.health - random(0, 3)
173.                                     organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(0, 3)
174.             case < 80:
175.                 organismFile[Id].stats.health =
organismFile[Id].stats.health - random(-2, 2)
176.                 organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(-2, 2)
177.             default:
178.                 organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate - random(0, 5) // Food is surplus, retention rate decreases
179.             endswitch
180.
181.             switch terrainAttributes.oxygenAvailability // if oxygen is scarce, harsh hit
on health, but gain in retention rate
182.                 case < 50:
183.                     organismFile[Id].stats.health =
organismFile[Id].stats.health - random(0, 10)
184.                     organismFile[Id].stats.retentionRate =
organismFile[Id].stats.retentionRate + random(0, 10)
185.                 default:
186.                     organismFile[Id].health[0].health =
organismFile[Id].stats.health + random(1, 5)
187.                 endswitch
188.
189.             switch organism[Id].stats.retentionRate // if the organism has a high retention
rate, they are able to negate the effects of reduced oxygen/water/food
190.                 case > 80:
191.                     organismFile[Id].stats.health =
organismFile[Id].stats.health + random(4, 15)
192.                 case > 60:
193.                     organismFile[Id].stats.health =
organismFile[Id].stats.health + random(3, 12)
194.                 case > 40:
195.                     organismFile[Id].stats.health =
organismFile[Id].stats.health + random(2, 10)
196.                 case > 20:
197.                     organismFile[Id].stats.health =
organismFile[Id].stats.health + random(1, 5)
198.                 default:
199.                     organismFile[Id].stats.health =
organismFile[Id].stats.health + random(0, 1)
200.             endswitch
201.
202.             if organismFile[Id].stats.health > 100 then
203.                 organismFile[Id].stats.health = 100
204.             endif
205.
206.             organismFile[Id].age = organismFile[Id].stats.age + 1 // adding 1 year to the
organisms age
207.         endprocedure
208.     endclass

```

pseudocode

This class is vital for the functioning of EvSim, as it allows for organisms to evolve. While this is a very early design of that, it demonstrates the principle of evolution within EvSim, and provides a building opportunity for further development. Within this, class, there are 2 procedures, a function, and the class to link them all. The procedure updateStats() updates the organism itself based on what terrain the organism is in. There is a lot of room for this to go further in depth, as well as attributes to be added, but this is just to primitively show the logic behind the procedure. updateOrganismFile(), is like the previous procedure, within which it updates the location and terrain for the organism so that it can be repainted appropriately. Both procedures are called through the main class, Organisms. Which runs continuously whilst the game is not paused. This class also calls the function makeDecision, which returns the new coordinates after the program makes a move, if it chooses to do so. In principle this file will control both the movement of organisms as well as the evolution itself.

2.3.4 Terrain Attributes

As seen in the previous section, the Organisms file must take information from an external class, within the file TerrainAttributes.java. Below is the pseudocode for that file.

```
1. // TerrainAttributes.java
2. // This is a basic class outlining the effect that each biome has on the organisms
3. // Used by Organisms.java for the control of the attributes
4. // There are more examples of biomes here to demonstrate the diversity between the different
biomes
5.
6. class baseAttributes // The base settings for all biomes
7.     str mainFoodSource = "Plants"
8.     float healthGain = 0.125
9.     float speedGain = 0.5
10.    int temperature = 20
11.    int oxygenAvailability = 100
12.    int foodAvailability = 50
13.    int waterAvailability = 50
14. endclass
15.
16. class grass inherits baseAttributes
17.     public terrainId = 0
18.     healthGain = healthGain + 0.2
19.     speedGain = speedGain + 1.5
20.     temperature = temperature - random(-5, 10)
21.     foodAvailability = 100
22.     waterAvailability = 50
23. endclass
24.
25. class water inherits baseAttributes
26.     public terrainId = 1
27.     mainFoodSource = "SeaLife"
28.     int alternateFoodSource = "Plants"
29.     healthGain = healthGain - 0.025
30.     speedGain = speedGain - 0.25
31.     temperature = temperature - random(7, 15)
32.     foodAvailability = 20
33.     waterAvailability = 100
34.     int alternateFoodSourceAvailability = 5
35.     oxygenAvailability = oxygenAvailability - 50
36. endclass
37.
38. class mountains inherits baseAttributes
39.     public terrainId = 2
40.     speedGain = speedGain - 0.45
41.     temperature = temperature - random(10, 20)
42.     foodAvailability = 40
43.     waterAvailability = 50
44. endclass
45.
46. class desert inherits baseAttributes
47.     public terrainId = 3
48.     int alternateFoodSource = "Animals"
49.     healthGain = healthGain - 0.325
50.     speedGain = speedGain - 0.5
51.     temperature = temperature + random(3, 20)
52.     foodAvailability = 10
53.     waterAvailability = 5
54.     int alternateFoodSourceAvailability = 2
55. endclass
56.
57. class snow inherits grass
58.     public terrainId = 4
59.     healthGain = healthGain - 0.1
60.     speedGain = speedGain - 1
61.     temperature = temperature - random(14, 25)
62.     foodAvailability = foodAvailability - 35
63.     waterAvailability = waterAvailability - 45
64. endclass
```

```

65.
66. class forest inherits grass
67.     public terrainId = 5
68.     healthGain = healthGain + 0.4
69.     speedGain = speedGain - 2.5
70.     foodAvailability = 100
71.     waterAvailability = 100
72. endclass
73.
74. class deepWater inherits water
75.     public terrainId = 6
76.     healthGain = healthGain - 0.025
77.     speedGain = speedGain - 0.5
78.     temperature = temperature - random(10, 20)
79.     oxygenAvailability = oxygenAvailability - 0.35
80.     foodAvailability = foodAvailability - 5
81.     alternateFoodSourceAvailability = alternateFoodSourceAvailability - 3
82. endclass

```

pseudocode

A seen in the pseudocode, each class represents what the factors affecting the organism are for each different biome. In this file there are more biomes than originally laid out, this is to demonstrate the biomes that are possible to be created within EvSim. Having each biome have a class also allows for the user to potentially change the attributes affecting the organism, further allowing customisation, a key factor of EvSim.

2.4 Usability Features

For a program as in depth as EvSim, the menus and features need to be clear and easily accessible to all. As I have previously mentioned, the UI within EvSim is only going to be simple and doesn't need to be complex at all. For example, the opening menu will have a basic burger style layout, with the option to start a new simulation, the option to load into an old one, general options for the program (key binds etc), and the option to quit the program. Keeping the menus and layouts simple is beneficial for both the front-end GUI, and the backend processing (two subgroups of EvSim, see 2.2). A burger style menu is used throughout both websites and games/apps, it is simplistic and allows users to select and change settings such as navigation and elements. A menu like this was also seen in [1.3.1 Evolution Simulator by Primer Learning](#), with a menu at the bottom left allowing the user to filter by attributes. For the opening menu, the four mentioned options are the only ones needed, with there not being anything else the program needs to do.

The opening menu is only one example of usability features within EvSim, another example being the ability to change the colour sample that the program shows, to help the user understand the program more. For example, once the simulation begins, the map will be displayed through colours, with each colour representing a different biome. This could represent a significant issue in the event that a user is colour blind. Due to the fact that each biome is recognisable singularly due to the different colours, someone who isn't able to see the colours will struggle to differentiate the biomes. Therefore EvSim will allow the user to define their own colours for each biome. There will of course be a default colour scheme, and the user does not have to change that, however it will be consistent with the idea of usability within EvSim and be in the best interest of the stakeholders to allow them to change colours.

Another example of useability in EvSim, is the menu once the simulation has started. Due to the simulation being completely fluid (open to change), the menu needs to be intuitive and simple to use. I plan to allow the user to change each organism, as well as make changes to the map. Therefore, it makes sense to have a side menu area where users will be able to edit the current state of the map and organisms. This side menu will be static on the side and will change depending on what the user is currently doing. For example, if the user clicks on an organism, they will be provided with the current stats (listed in 1.4 essential features), and a button to edit those. If the user hasn't clicked on an organism, the program will display all the current stats and global averages (number of organisms, total

organism count etc). This means that whether or not the user has selected to view a particular organism, they are able to view the process of EvSim, and therefore the evolution itself, matching with 1.7 Success Criteria. Keeping the design of the side menu self-intuitive means there is less for the end user to learn, therefore it is likely they are going to use EvSim to its full potential.

2.5 Variables and Validation

Within this section, I will cover the majority of variables and data structures within the EvSim project, as well as outline why they are necessary, and what they provide to the system. I will also cover why they are defined as they are – why it is a particular variable. This will provide a deeper insight into the design for EvSim. Similar to 2.3, I will cover it file by file, starting with the map generation.

2.5.1 Map Generator

Name	Data Type	Scope	Purpose?
WIDTH	Integer	Public	Defines the width of the game panel on screen, can be changed as needed, or even by the user. Used to set the number of columns to be generated on the world map. Integer as needed for integer division. Can be changed by the user, or read by the game panel itself, therefore needs to be public
HEIGHT	Integer	Public	Defines the height of the game panel on screen, can be changed as needed, or even by the user. Used to set the number of rows to be generated on the world map. Integer as needed for integer division. Can be changed by the user, or read by the game panel itself, therefore needs to be public
cellSize	Integer	Private to MapGenerator	The size of the cells, each cell is a 5 * 5 size, each organism can move over a 1 * 1 size. This increases the amount the organism needs to travel overtime. Integer as needed for integer multiplication. Only needed to generate the terrain size within MapGenerator, and therefore is private.
numberOfRows	Integer	Private to MapGenerator	The number of rows to generate, each row is cellSize width. Integer as needed to be iterated through. Private as only needed within MapGenerator.
numberOfCols	Integer	Private to MapGenerator	The number of columns to generate, each row is cellSize height. Integer as needed to be iterated through. Private as only needed within MapGenerator.
map	Integer array	Public	Integer array defining the terrainType of each cell. Has two uses -> Coordinate grid, defines the biome. This array is necessary for linking where the organism is, to what terrain it is on. The terrainType can be returned (as an integer), by a function fetching the

			integer at position [x][y]. Public as is needed throughout the project.
terrainType	Integer	Private to generateWorldMap	Defines the different biomes. Integer to allow to be passed through the map integer array. Also allows for easy transitioning of biome, as well as adding, or taking away what biomes are available. Private as only needs to be used for the initial map generation within generateWorldMap.
grass	Integer	Private to generateWorldMap	One of the biomes that can be generated. Biomes can be added or changed by the user. Integer to be parsed through terrainType into map[][]]. Private as only used within generateWorldMap.
water	Integer	Private to generateWorldMap	One of the biomes that can be generated. Biomes can be added or changed by the user. Integer to be parsed through terrainType into map[][]]. Private as only used within generateWorldMap.
chance	Integer	Private to generateWorldMap	The random chance defining which biome is to be spawned, this can be changed to make one biome spawn more, or changed to different percentages to add another biome. Integer as needs to be compared. Private as only used within generateWorldMap.

2.5.2 Organism Generation and movement

2.5.2.1 *OrganismGenerator.java*

Name	Data Type	Scope	Purpose
organismCount	Integer	Private to OrganismGenerator	The number of organisms to generate. Can be defined by the user. Is iterated through to create the amount of organisms and therefore is an integer. Only needs to be used within OrganismGenerator and is therefore private.
organismRadius	Integer	Private to OrganismGenerator	The size of each organism within the program. Can also be changed by the user if needed. Integer as it represents the size of the organism to paint. Private as only needed within OrganismGenerator.

X and Y	Integer	Private to OrganismGenerator.	Consistently updated and sent to organismFile as the organisms spawning coordinates.
organismFile	JSON	Public, restricted to GenerateOrganisms procedure.	The file containing all the organism information. Is already a blank file within the project directory, therefore does not need to be created and is only used in this instance within GenerateOrganisms procedure.

Organisms.json – The variables defined into this file.

Name	Data Type (if applicable)	Initial value (if applicable)	Purpose
organismId	Integer	0	Used to differentiate between organisms. Linking the organism to its correct attributes. Defining as an integer is the most efficient way to differentiate organisms. Starts at 0, and ends at organismCount (see above)
x-coordinate	Integer	x (see above)	The current x coordinate for the organism. This will change frequently when the organism moves (see 2.5.2.2). Defined as an integer to be painted on the correct column in map[][].
y-coordinate	Integer	y (see above)	The current y coordinate for the organism. This will change frequently when the organism moves (see 2.5.2.2). Defined as an integer to be painted on the correct row in map[][].
currentTerrain	Integer	The terrain at [x][y], uses getTerrain method.	Uses the getTerrain method to return what the current terrain is at x (global integer – see above) and y (global integer – see above). This is read and updated when the organism is processed, to change the stats accordingly.
stats	Object		This contains all the stats for the organism. Having the stats in an object of its own creates a more readable JSON.

currentForm	Integer	0	Is read when it is painted to the map, can change depending on the environment. Integer to represent different shapes. This can then be easily changed by the user if needed.
retentionRate	Integer	50	A stat for the organism. Is processed and updated in the OrganismAttributes file, see 2.5.3.
age	Integer	0	"
health	Integer	100	"
speed	Integer	1	"
awareness	Integer	0	"
size	Integer	10	"
attack	Integer	0	"
defence	Integer	0	"
preferredTemperature	Integer		"
preferredFood			"
preferredTerrain	Integer		"
isDead	Boolean	false	"

2.5.2.2 Organism Movement

Name	Data Type	Scope	Purpose
currentOrganismId	Integer	Public, restricted to the procedure updateOrganism	Used to verify the correct organism is being updated. Used in a for loop to verify which organism to update. Integer as checked against another integer in line 9.
newX	Integer	Parsed into the procedure updateOrganism when it is called.	The new x-coordinate that is generated. Updates what is stored in organisms.json to be repainted/read again.
newY	Integer	Parsed into the procedure updateOrganism when it is called.	The new y-coordinate that is generated. Updates what is stored in organisms.json to be repainted/read again.

2.5.3 Organism Attributes

This is a large file, therefore it will be split down into the separate procedures.

public class Organism			
Name	Data Type	Scope	Purpose

Organism	class	Public, needs to be accessible to be called in other files.	Hold the procedures and functions which cause the organisms to update one by one. Contains the updating process for both stats and positioning. Important for the execution of EvSim, as it is the process of Evolution.
Id	Integer	Private, but accessible to all procedures and functions within this file.	The current organism's id that is being operated on. Verifies that the current organism is being updated correctly. Needs to be accessible to the whole file as it is set in Organisms(), and read in makeDecision, updatestats, and updateOrganismFile.
public procedure Organisms			
Name	Data Type/Structure	Scope	Purpose
Organisms	Procedure	Public	The main procedure, iterates through each organism repeatedly, calls the function makeDecision, and procedures updateStats and updateOrganismFile. Causes the organisms to move and update. Needs to be public so it can be called externally when the game is in running.
preferredTerrain	Integer	Private	Allows the program to check whether the organism is currently in the preferred biome. If it is a new organism, the preferredTerrain which is stored is null, and is therefore set to the terrain within which it is currently in. is sent to makeDecision, therefore is private to Organisms (procedure). Needs to be an Integer to be set from organisms.json.
currentTerrain	Integer	Public	Fetches the current terrain from organisms.json, and sends it to makeDecision so the function can move the organism. Moved between functions, therefore needs to be public. Integer to be compared and set as the integer in organisms.json.
currentX	Integer	Public	The current x-coordinate of the organism, is updated in makeDecision, and used in updateOrganismFile, therefore needs to be public.
currentY	Integer	Public	The current y-coordinate of the organism, is updated in makeDecision, and used in updateOrganismFile. Therefore needs to be public.
newXY	Array	Private	Calls the function makeDecision, and takes the new x and y coordinates that are returned from the function. Declared as an array so that both x and y coordinates can be returned. X coordinate is in index 0 of

			array, and y coordinate is in index 1 of array. Only ever used within Organisms.
Private function makeDecision(preferredTerrain: byVal, currentTerrain: byVal)			
Name	Data Type/Structure	Scope	Purpose
makeDecision	Function	Private	Takes the organisms current terrain and preferred terrain, compares, and causes the organism to move, if it is not the preferred terrain. Small chance to move if it is the preferred terrain so the organism isn't static. Function as it returns the new x and y coordinates. Private as only needs to be used and called by Organisms.
preferredTerrain	Integer	Private	The organisms preferred terrain, compared against currentTerrain to determine if the organism should move. Integer to match the terrain types defined in 2.5.1. Private as only used within this function.
currentTerrain	Integer	Private	The organisms current terrain, compared against preferredTerrain to determine if the organism should move. Integer to match the terrain types defined in 2.5.1. Private as only used within this function.
newX	Integer	Private	The new x-coordinate that is generated for the organism to move to. Integer as it represents the place on the grid. Private as is only used within the function and then returned at the end.
newY	Integer	Private	The new y-coordinate for the organism to move to. Integer to represent the place on the coordinate grid. Private as only used within this function and returned at the end.
newXY	Array	Private	Used to return both newX and newY, returns as an array so x and y coordinate can be differentiated. Private as only declared and used within this function.
Private procedure updateStats(int Id: byVal)			
Name	Data Type	Scope	Purpose
updateStats	Procedure	Private	Updates the stats of the organism based off of the current terrain's attributes (see 2.5.4). procedure as nothing is returned, it only updates the organsimFile. Private as only used within this class.
terrainAttributes	Object	Private	Fetches the current terrain's attributes as an object, so the individual attributes can be processed.

2.5.4 Terrain Attributes

Name	Data Type	Scope	Purpose
baseAttributes	Class	Public – Needs to be accessible by other	Defines the base attributes for all biomes. Declares the base

		classes and files so calculations can take place	variables that change per biome, explained below.
mainFoodsource	String	"	The main food source of an organism. As a string so it is in a human readable form. Can still be compared.
healthGain	Float	"	The health that is gained within that specific terrain. Is a float as these values are very small.
speedGain	Float	"	The speed that is gained within the biome, is once again small, therefore a float.
temperature	Integer	"	The temperature of the biome, only measured in whole numbers, therefore an integer.
oxygenAvailability	Integer	"	The amount of resource available to the organism. Stored and used as a percentage therefore is an integer.
foodAvailability	Integer	"	"
waterAvailability	Integer	"	"
terrainId	Integer	"	The terrain id, links to the biomes that will be painted in 2.5.1/2.3.1.
grass	Class	"	A biome class, which takes the base attributes from baseAttributes, and changes them depending on the specific attributes of that terrain.
water	Class	"	"
mountains	Class	"	"
desert	Class	"	"
snow	Class	"	"
forest	Class	"	"
deepWater	Class	"	"
alternateFoodSource	String	"	The amount of resource available to the organism. Stored and used as a percentage therefore is an integer.
alternateFoodSourceAvailability	Integer	"	"

2.6 Iterative Test Data

Test data for MapGenerator			
Data	Expected	Boundary	Erroneous
TerrainType	0, 1	0, 1	-1 >=, 2 <=

			There are only 2 terrain types that can be made, grass (0), and water (1). Anything other than these two types are not expected.
Map[][]	[1][318], [197][104], [216][4]	[216][384], [0][0]	[1120][500], [905][2001], [1520][-1] There should only be values up to 1080 / cellSize, and 1920 / cellSize. cellSize is set at 5, therefore 216 x 384. Any negative values, or values larger than 216 x 384 are not expected.
Test data for OrganismGenerator			
x	225, 354, 181, 2	0, 540	< 0, > 540 Can only be within the boundary 0, HEIGHT / organismRadius = 540 => 0, 540. Anything else is not expected.
y	96, 52, 882, 458	0, 960	< 0, > 540 Can only be within the boundary 0, WIDTH / organismRadius = 960 => 0, 960. Anything else is not expected.
organismFile.currentTerrain	0, 1	0, 1	< 0, > 1 There are only 2 terrain types that can be within the map, therefore anything other than that are not expected.
Test data for OrganismGenerator			
newX	125, 5, 540	0, 540	< 0, > 540 Can only be within the boundary 0, 540. Anything else is not expected.
newY	125, 2, 925	0, 960	< 0, > 960 Can only be within the boundary 0, 960. Anything else is not expected.

Test data for Organisms.java is slightly different, as the main test here will be for the organisms statistics. Therefore we will test for the statistics that are generated on each organism.

Test data for organism statistics		
Typical	Boundary	Erroneous
Age: 5 Health: 82 Speed: 4 Awareness: 0 Size: 1 Attack: 0 Defence: 0 preferredTemperature: 22 preferredFood: "Plants"	Age: 0 Health: 1 Speed: 0.1 Awareness: 0 Size: 10 Attack: 10 Defence: 10 preferredTemperature: 40 preferredFood: "Sealife"	Age: -1 Health: 101 Speed: 0 Awareness: -1 Size: 102 Attack: -1 Defence: -1 preferredTemperature: 105 preferredFood: 1
Age is purely an integer that increases overtime, therefore it cannot be a negative integer. Health is bound to 0 and 100, once it reaches 100, the organism is declared as dead. Speed is also bound to >0, as an organism cannot be immovable. Realistically to limit movement, the organism cannot also have a speed of greater than 10. Awareness, attack, and defence are also limited to 10, but can also be a value of 0. Size is restricted to 10, as an organism should not be too large to not cause discrepancies in the game. Preferred temperature for an organism can only be the same as what the different terrains are. Preferred food is shown as a string, therefore cannot be represented as an integer.		

These tests are a few examples of tests and test data which will be used during development, while certain factors may not be replicated (due to unforeseen changes), it demonstrates what tests will be carried out throughout the development process. Please refer to specific sections labelled 'Testing' for the tests themselves. 3.1.1.2 for example.

2.7 Post-development Test Data

While a lot of information regarding what will be tested post-development is not yet known, I have planned for tests which may be applicable to EvSim, for example, the map generation characteristics.

Test #	Description	Test Data	Expected Outcome
1	The user should be able to define the number of organisms to display before starting the simulation.	5 Organisms	5 Organisms should be shown and displayed to the user once the simulation has started.
2	The user should be able to define the map size before starting the simulation.	Map seed of 192. Screen size of 850 x 700	The map shown in a window size of 850 x 700.
3	The map should have the ability to be customised by the user, through a 'seed' input	A random number within the map seed range (undefined)	The random map seed should be generated and generated the same each time it is generated.
4	The layout of the map should be consistent with what is seen in real life	A random number within the map seed range (undefined)	The map should follow the following layout in order of contact: deep water, water, beach, grass, forest, grass, mountain, snow.
5	The save simulation function should save the current game state.	The current simulation	The current simulation should be saved its state, and be able to be opened again in the same state.
6	The data displayed in the side menu should replicate what is shown to the user.	A random simulation	The data shown on the side menu for a selected organism should replicate what is displayed visually to the user, for example the size of an organism.
7	Organisms will be tested to ensure that evolution is correctly displayed.	Size/Colour of organisms should represent their actual genes.	The key which displays organism appearance should match organisms within the simulation with similar attributes.
8	Organisms should reproduce to create a child organism	New simulation.	2 Organisms going within proximity which are opposite gender will create a child
9	The program operating with changing parameters. This is the visual display to the	A new simulation.	The simulation should update the organisms frequently over a period

	user. Any changes such as movement should be shown frequently and accurately.		of 50 years. Organisms should move about the map, change in size, and give birth to other organisms.
10	The user's ability to search for an organism should be swift and accurate.	A new simulation. Random organism.	The organism ID which was searched for should return the linked organism and its attributes. The organism should be highlighted on the screen, and the attributes displayed on the side menu.
11	This tests the family tree of an organism, checking that the attributes are similar.	An aged simulation. One organism will be chosen that is currently alive. One organism and their ancestors will be compared.	The family tree of the organism should follow in line with the current attributes of the organism. For example, the size of the organism should not differ far from the parents' sizes.
12	The main menu should lead the user into starting a new simulation, loading into an old one, change the appearance of EvSim, and quit the game.	N/A	All buttons on the main menu should function with the intentional outcome. The 'Start' button should lead the user into the selection menu. The 'Load' button should lead the user to select from a list of previous simulations. The 'Options' button should lead the user to select their colour and window size options. The 'Exit' button should allow the user to quit the program, ending its operation.
13	The application should allow the user to customise the apps appearance. The colourblind accessibility options should display to the user the change that it will make, allow the user to save that combination, and make the change have affect.	N/A	Any changes made to specific terrains should be previewed on the menu, with any changes saved being kept within the program, even after restarting.
14	The map should be painted in the same layout as the users selected colour scheme.	N/A	The colours selected and saved from test 2 should be replicated on the actual map which is displayed to the user

			once the simulation is started.
15	The side menu should allow the user to speed up, slow down, and pause the simulation.	N/A	The speed up, slow down, and pause game buttons should carry out their respective effect on the simulation.
16	The side menu should allow the user to view specific organisms, and global attributes.	N/A	The Find Organism tab should allow the user to input an organism ID and select find. If the organism is found then the side menu should display the organisms statistics, and if not it should display an appropriate error message.

3.0 Development

3.1 Front End UI

3.1.1 Display of the Map

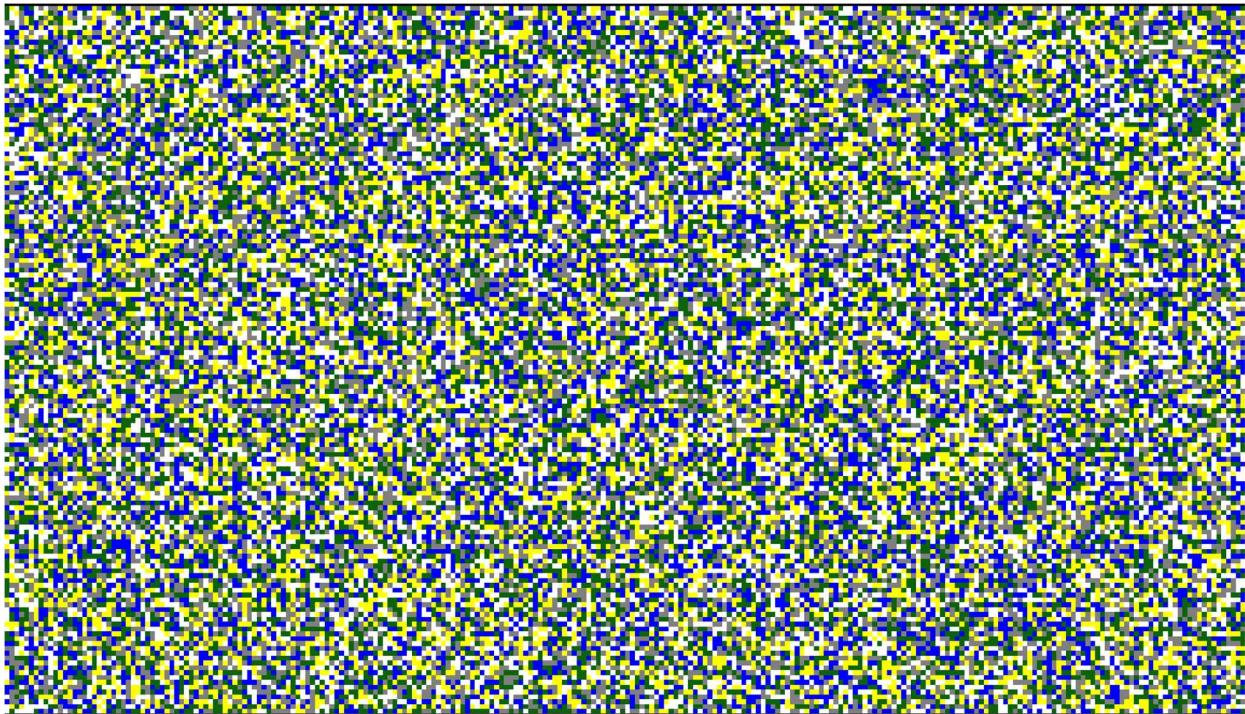
During early development stages of the map, the algorithm which was designed in 2.3.1 was not found to be accurate nor suitable for the project. The algorithm below follows along with the design (see 2.3.1), with a photo of what it displays.

```

1.     private void generateWorldMap() {
2.         int GRASS = 0, WATER = 1, MOUNTAINS = 2, DESERT = 3, SNOW = 4, FOREST = 5;
3.         int chance;
4.         for (int row = 0; row < NUM_ROWS; row++) {
5.             for (int col = 0; col < NUM_COLS; col++) {
6.                 int terrainType = GRASS; // Default to grass
7.
8.                 chance = random.nextInt(0, 100);
9.
10.                if (chance < 20) {
11.                    terrainType = WATER;
12.                } else if (chance < 40) {
13.                    terrainType = MOUNTAINS;
14.                } else if (chance < 60) {
15.                    terrainType = DESERT;
16.                } else if (chance < 80) {
17.                    terrainType = SNOW;
18.                } else if (chance < 100) {
19.                    terrainType = FOREST;
20.                }
21.                map[row][col] = terrainType;
22.            }
23.        }
24.    }

```

Java



This was unsuitable for EvSim, due to it not being representative of a realistic map. Further testing with random chance led to no improvements. A decision was made to instead use [Perlin Noise](#) instead. Noise functions allow for a map that is bunched together in a pattern, more like what a proper map would look like. Noise functions such as Perlin noise are used in 2d and 3d games like Minecraft to randomly generate world maps that contain realistic elements. I intend to use Perlin Noise in a similar way.

3.1.1.1 Implementation of Perlin Noise

Using the assistance of Perlin Noise [documentation\[1\]](#), a [Wikipedia article](#), and an example from [GitHub\[2\]](#), I was able to implement a basic map into test instance. Perlin Noise is commonly used in other games to generate world maps, a key example of this is Minecraft. I am using Perlin Noise for a similar implementation, as it is the closest solution to what I am trying to achieve. The code below is a primitive version of my implementation, one which needs more refining and tests.

```
1.     public void generateWorldMap(){
2.         double scale = 500.0; // The larger this value, the larger the map appears (the more
noise) - Also causes performance decrease.
3.         perlinNoiseTest noiseGenerator = new perlinNoiseTest(); // Bringing in the noise
function
4.         for (int x = 0; x < NUM_ROWS; x++) {
5.             for (int y = 0; y < NUM_COLS; y++) {
6.                 // Generates noise
7.                 double nx = (double) x / NUM_COLS;
8.                 double ny = (double) y / NUM_ROWS;
9.                 double terrainValue = noiseGenerator.noise(nx * scale, ny * scale);
10.                double normalisedValue = (terrainValue + 1.0) / 2.0; // This changes the value
to be between 0 and 1
11.                map[x][y] = normalisedValue; // Sending the noise value to a double (float)
array.
12.            }
13.        }
14.    }
15.
16.    @Override
17.    protected void paintComponent(Graphics g) {
18.        super.paintComponent(g);
19.
20.        // Defining colours for terrain types
```

```

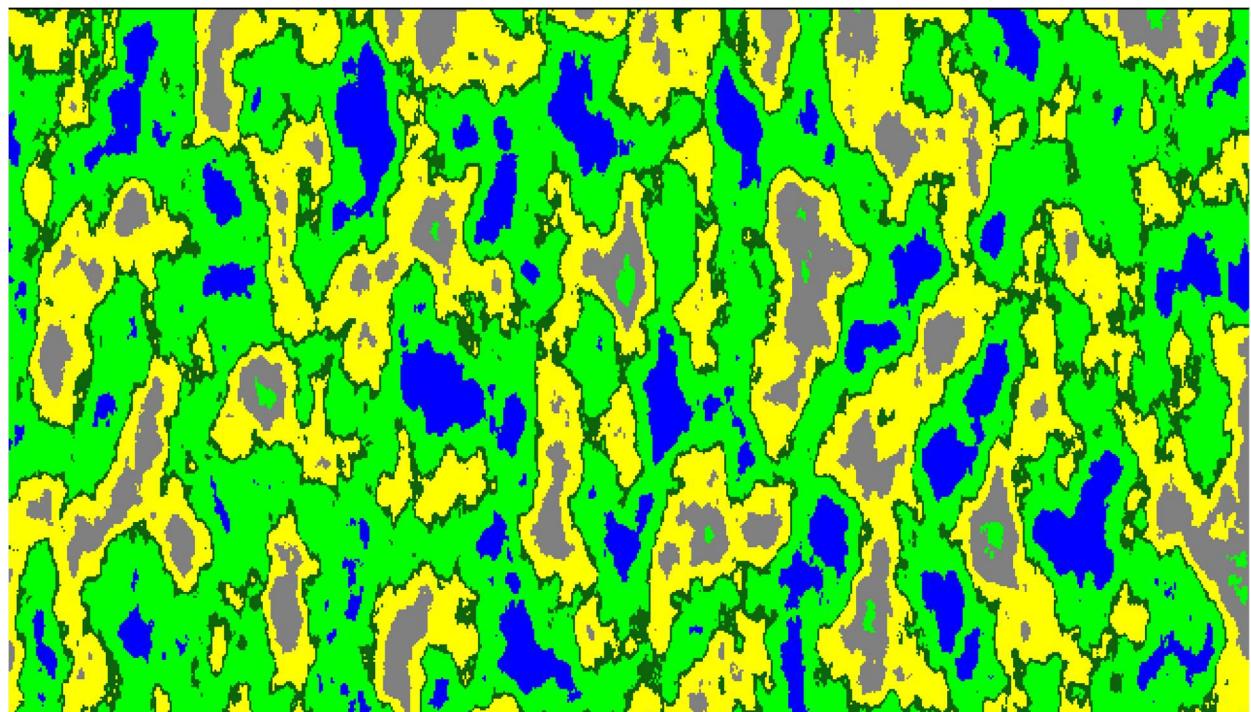
21.         Color[] terrainColors = {Color.GREEN, Color.BLUE, Color.GRAY, Color.YELLOW,
22. Color.WHITE, new Color(13, 100, 10) /* dark green */ };
23.         for (int row = 0; row < NUM_ROWS; row++) { // This loop sets the painted colour to be
representative of what biome it should be.
24.             for (int col = 0; col < NUM_COLS; col++) {
25.                 double terrainType = map[row][col];
26.                 Color terrainColour;
27.                 if(terrainType < 0.3){
28.                     terrainColour = terrainColors[1]; // The lower the noise, the 'lower' the
terrain, therefore the closer to 1, the higher the map should be. In this instance, values under
0.3 is shown as water.
29.                 } else if (terrainType < 0.4) {
30.                     terrainColour = terrainColors[0]; // Values under 0.4 will be grass
31.                 } else if (terrainType < 0.55) {
32.                     terrainColour = terrainColors[5]; // Values under 0.55 will be forest
33.                 } else if (terrainType < 0.7) {
34.                     terrainColour = terrainColors[3]; // Values under 0.7 will be desert
35.                 } else if (terrainType < 0.9) {
36.                     terrainColour = terrainColors[2]; // Values under 0.9 will be mountains
37.                 } else terrainColour = terrainColors[0]; // Any other value default to grass
38.
39.                 g.setColor(terrainColour);
40.                 g.fillRect(col * CELL_SIZE, row * CELL_SIZE, CELL_SIZE, CELL_SIZE); // Draws
the biome onto the window.
41.             }
42.         }
43.     }

```

Java

The Perlin noise code itself, whilst it is pre-programmed, I have adapted certain methods to better suit my purposes, more specifically for the seed setting and fetching. To see how Perlin noise was implemented into my program, see code reference C1.9.

Below is a test demonstration for what the code above produces.



As demonstrated in the above image, the terrain generation here is not perfect, as there are multiple discrepancies. Values will have to be tested repeatedly in order to generate a smoother map. For example, an issue that was noticed straight away, was line 37, where the default value (higher values) was set to grass. As seen in the image, this presents an issue, where at the highest parts of the a mountain biome, there was grass. In this case, I would need to change

```

else terrainColour = terrainColors[0]; // 0 represents the grass biome (in the terrainColors
object)
to
else terrainColour = terrainColors[4]; // 4 represents the snow biome - which would generate the
snow biome instead.

```

3.1.1.2 Testing Perlin Noise

This section will be going through this test implementation to create a final form of Perlin Noise which I can use to generate random world maps.

The first major improvement I could make to the code was to change the areas within which biomes spawn. For example in the example in 3.1.1.1, many of the biomes were misrepresented, or at different scales. A prime example of this is the desert biome, which was seen in great amounts between the forest and the mountain biomes, an unnatural place for deserts to spawn. This was fixed with a simple swap of grass and desert, as well as changing the chance of desert spawning. The ‘desert’ biome, now more suits a ‘beach’ biome, meaning earlier defined terrain attributes will have to change. Similarly, the forest in the first iteration was only a small section along the edge of the desert. A major improvement would be to expand this forest, and move it to the middle of any grass biomes, to make the map look and function better. To achieve this, I would need to define values for the forest between grass. For example grass would spawn between 0.6 and 0.8, with forest between 0.65 and 0.7. As mentioned in the final part of 3.1.1.1, I would also need to change line 37 to snow, instead of grass.

After going through and testing different values and variables for generation, this is the final version of Perlin Noise for map design. The code below is the final version for how the different biomes are to spawn, as well as 2 examples of what this looks like at different scales.

```

1.    @Override
2.    protected void paintComponent(Graphics g) {
3.        super.paintComponent(g);
4.
5.        // Define colors for terrain types
6.        Color[] terrainColors = {Color.GREEN, Color.BLUE, Color.GRAY, Color.YELLOW,
Color.WHITE, new Color(13, 100, 10) /* dark green */, new Color(6, 11, 112) /* Dark Blue */ };
7.
8.        for (int row = 0; row < NUM_ROWS; row++) {
9.            for (int col = 0; col < NUM_COLS; col++) {
10.                double terrainType = map[row][col]; // Intialising a new 2d array
11.                Color terrainColor = getColor(terrainType, terrainColors);
12.                // Fetching and setting the colour to paint
13.                g.setColor(terrainColor);
14.                g.fillRect(col * CELL_SIZE, row * CELL_SIZE, CELL_SIZE, CELL_SIZE); // Paints
as a square on the screen at position col * CELL_SIZE, row * CELL_SIZE
15.            }
16.        }
17.    }
18.
19.    private static Color getColor(double terrainType, Color[] terrainColors) {
20.        Color terrainColor;
21.        if(terrainType < 0.3) {
22.            terrainColor = terrainColors[6]; // Perlin Noise can only generate numbers between
0 and 1, anything that is generated below 0.3 will be deep water.
23.        } else if(terrainType < 0.5) {
24.            terrainColor = terrainColors[1]; // Anything below 0.5 will be water
25.        } else if (terrainType < 0.55) {
26.            terrainColor = terrainColors[3]; // Anything below 0.55 will be desert
27.        } else if (terrainType < 0.6) {
28.            terrainColor = terrainColors[0]; // Anything below 0.6 will be grass
29.        } else if (terrainType < 0.65) {
30.            terrainColor = terrainColors[5]; // Anything below 0.65 Will be forest
31.        } else if (terrainType < 0.7) {

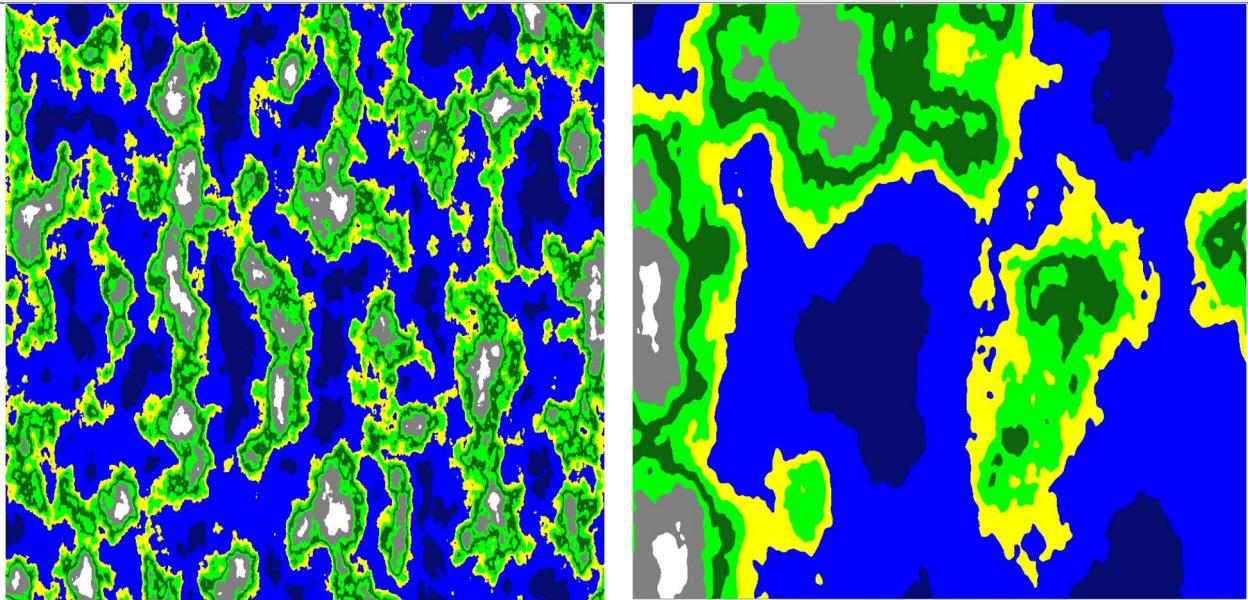
```

```

32.         terrainColor = terrainColors[0]; // Anything below 0.7 will be grass
33.     } else if (terrainType < 0.8) {
34.         terrainColor = terrainColors[2]; // Anything below 0.8 Will be mountains
35.     } else terrainColor = terrainColors[4]; // Anything above that will then be snow
36.     return terrainColor;
37. }

```

Java



Scale 500 (left), scale 100 (right). Depending on how large the user wants the map, they are able to change the scale to represent this, the larger the scale, the larger the map. There will be a list of defined values for the user to choose from to ensure usability.

3.1.1.3 Final code

The final section is the full section of code which is called when the map is generated within the game.

```

1. import javax.swing.*;
2. import java.awt.*;
3.
4. public class generateMap extends JPanel {
5.     public static int WIDTH = 1920; // Default window size, can be changed
6.     public static int HEIGHT = 1080;
7.     public double scale = 100; // This can be changed / adjusted to change the size of the map
that is generated
8.     public static double[][] map;
9.
10.    public generateMap() {
11.        map = new double[HEIGHT][WIDTH];
12.        perlinNoise noiseGenerator = new perlinNoise(); // Initialises the perlin noise
generator
13.        for (int x = 0; x < HEIGHT; x++) {
14.            for (int y = 0; y < WIDTH; y++) {
15.                // Casts coordinates to generate noise
16.                double nx = (double) x / WIDTH;
17.                double ny = (double) y / HEIGHT;
18.                double terrainValue = noiseGenerator.noise(nx * scale, ny * scale);
19.                double normalisedValue = (terrainValue + 1.0) / 2.0; // Makes the generated
value between 0 and 1
20.                map[x][y] = normalisedValue;
21.            }
22.        }
23.    }
24.    public void paintComponent(Graphics g) {
25.        super.paintComponent(g);
26.        // Define colors for terrain types
27.        Color[] terrainColors = {Color.GREEN, Color.BLUE, Color.GRAY, Color.YELLOW,
Color.WHITE, new Color(13, 100, 10) /* DARK GREEN */, new Color(6, 11, 112) /* DARK BLUE */};

```

```

28.
29.         for (int row = 0; row < HEIGHT; row++) {
30.             for (int col = 0; col < WIDTH; col++) {
31.                 double terrainType = map[row][col];
32.                 Color terrainColor = getColor(terrainType, terrainColors); // Sets the correct
colour based on the value generated from noise at coordinates row, col
33.
34.                 g.setColor(terrainColor);
35.                 g.fillRect(col, row, 1, 1); // Paints them to the screen
36.             }
37.         }
38.     }
39.     private static Color getColor(double terrainType, Color[] terrainColors) { // Defines how
the biomes are generated.
40.         Color terrainColor;
41.         if (terrainType < 0.3) {
42.             terrainColor = terrainColors[6]; // Perlin noise can only generate numbers between
0 and 1, anything that is generated below 0.3 will be deep water.
43.         } else if (terrainType < 0.5) {
44.             terrainColor = terrainColors[1]; // Anything below 0.5 will be water
45.         } else if (terrainType < 0.55) {
46.             terrainColor = terrainColors[3]; // Anything below 0.55 will be desert
47.         } else if (terrainType < 0.6) {
48.             terrainColor = terrainColors[0]; // Anything below 0.6 will be grass
49.         } else if (terrainType < 0.65) {
50.             terrainColor = terrainColors[5]; // Anything below 0.65 will be forest
51.         } else if (terrainType < 0.7) {
52.             terrainColor = terrainColors[0]; // Anything below 0.7 will be grass
53.         } else if (terrainType < 0.8) {
54.             terrainColor = terrainColors[2]; // Anything below 0.8 will be mountains
55.         } else terrainColor = terrainColors[4]; // Anything above that will be snow
56.     return terrainColor;
57. }
58. }
59.

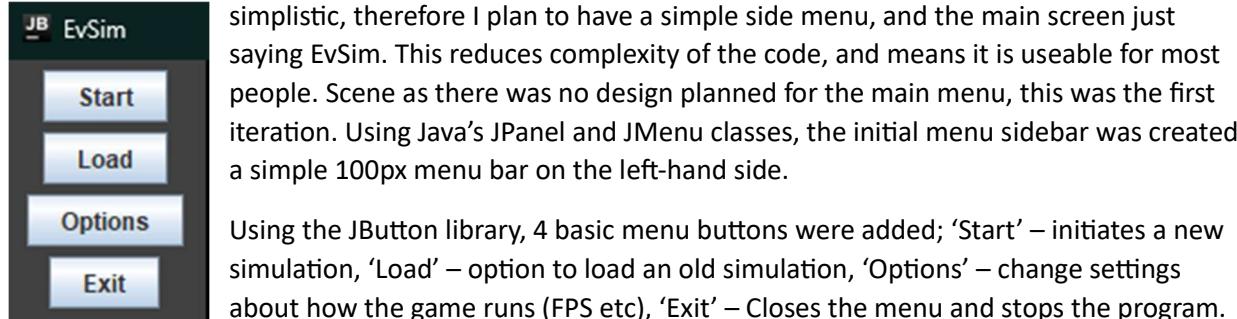
```

3.1.2 Opening Menu

3.1.2.1 Menu bar and buttons

Following along with what was planned in design regarding the opening menu, it only needs to be very

simplistic, therefore I plan to have a simple side menu, and the main screen just saying EvSim. This reduces complexity of the code, and means it is useable for most people. Scene as there was no design planned for the main menu, this was the first iteration. Using Java's JPanel and JMenu classes, the initial menu sidebar was created, a simple 100px menu bar on the left-hand side.



Using the JButton library, 4 basic menu buttons were added; 'Start' – initiates a new simulation, 'Load' – option to load an old simulation, 'Options' – change settings about how the game runs (FPS etc), 'Exit' – Closes the menu and stops the program.

```

1.     private JPanel createVerticalMenu() {
2.         JPanel menu = new JPanel();
3.         menu.setBackground(Color.darkGray);
4.         menu.setPreferredSize(new Dimension(100, getHeight())); // Sets the width to 100px and
the height to the same as the page
5.
6.         JButton startButton = new JButton("Start"); // Creates a new JButton
7.         JButton loadButton = new JButton("Load");
8.         JButton optionButton = new JButton("Options");
9.         JButton exitButton = new JButton("Exit");
10.        menu.add(startButton); // Adds the buttons to the side menu
11.        menu.add(loadButton);
12.        menu.add(optionButton);
13.        menu.add(exitButton);
14.    return menu;
15. }

```

Java

Certain elements of these buttons needed changing – thankfully JButton has in built functions to change the button appearance:

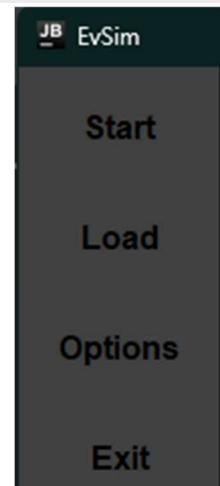
```
1.    private static void setButtonProperties(JButton button, String tooltip) { // Generic menu
button style
2.        button.setForeground(Color.black); // Sets the text colour to be black
3.        button.setBackground(Color.darkGray); // Sets the background button colour to be dark
gray (same as the menu)
4.        button.setToolTipText(tooltip); // Sets the help text that displays on mouse hover
5.        button.setPreferredSize(new Dimension(100, 50)); // Sets the button size
6.        button.setBorder(BorderFactory.createLineBorder(Color.darkGray, 1)); // Button border's
colour on click
7.        button.setFont(new Font("Arial", Font.BOLD, 16)); // The font and size of the text
8.    }
```

Java

```
1.    JButton startButton = new JButton("Start"); // Creates a new JButton
2.    JButton loadButton = new JButton("Load");
3.    JButton optionButton = new JButton("Options");
4.    JButton exitButton = new JButton("Exit");
5.
6.    // Sets the styling to the button
7.    setButtonProperties(startButton, "Starts a new iteration");
8.    setButtonProperties(loadButton, "Loads into a previous save");
9.    setButtonProperties(optionButton, "Set options for EvSim");
10.   setButtonProperties(exitButton, "Exit the program");
```

Java

Each button is now styled correctly and can be changed easily at any stage. The menu is now created, but not functional.



3.1.2.2 Adding functionality to the buttons

Adding action listeners to the buttons, enabled me to give the buttons functionality. At the current stage of development, I can only make the Start and Exit buttons functional. The other buttons will be added in a later section. Following along with the [structure of the solution \(see 2.2\)](#), I have decided to separate the options menu, and the actual game window to allow the user to separate the game from the menu. As defined in 2.2, the user needs to be able to start a new iteration and define the settings for the new iteration. To increase usability, it would make sense to pass these between separate windows, as it would be clearer to the user what the program wants them to do. Many websites and other games use pop out menus/modals to emphasise the direction that the program is taking, to ensure the user has the chance to change the settings available.

The start button uses the ActionListener() class to watch and react to a mouse click. In this instance, when the button is pressed, a new game window is generated, and the old opening menu is removed.

```
1.    startButton.addActionListener(new ActionListener() { // Creates and adds an action
listener from the ActionListener class
2.        @Override
3.        public void actionPerformed(ActionEvent actionEvent) {
4.            GameWindow gameWindow = new GameWindow(getWidth(), getHeight()); // Creates a
new game window from the game window class, same width and height as the current menu
5.            gameWindow.startGameThread();
6.            frame.dispose(); // Removes the current window that is displaying
7.        }
8.    });
```

Java

The GameWindow class currently loads the previous section, 3.1.1 – the display of the map. Under testing, the start button works successfully and efficiently at showing the map when the user clicks on it.

The exit button is much simpler, however also uses the Action Listener class. For the exit button to close the program, I have just used the inbuilt function `System.exit(0)` which causes the program to end.

```
1.     exitButton.addActionListener(new ActionListener() { // Creates and adds an action
2.         listener from the ActionListener class
3.         @Override
4.         public void actionPerformed(ActionEvent actionEvent) {
5.             System.exit(0);
6.         });

```

Java

Upon testing, this also works efficiently at closing the program when clicked.

3.1.2.3 Additional menu features

While the current menu works, it is not intuitive, and has a lot of empty space. The solution to this is once again simple, being functional and instructional for the user.

Using the `paintComponent` feature of `JPanel`, I added the text “EVSIM”, and “Press ‘Start’ to begin”, to the main menu. This would add simple instructions for the user, if they were unsure how to use the program. This text needed to be central to the menu, and clearly visible for the user to see. The initial code looked like this.

```
1.     @Override
2.     protected void paintComponent(Graphics g) { // This class 'paints' the contents to the
frame.
3.         super.paintComponent(g);
4.
5.         g.setColor(Color.white); // Sets the text colour to white
6.
7.         g.setFont(new Font("Agency FB", Font.BOLD, 108));
8.         g.drawString("EVSIM", 860, 540); // Adds the string "EVSIM" to the center of the screen
9.
10.        g.setFont(new Font("Agency FB", Font.BOLD, 36)); // Changes to a smaller font for the
instruction
11.        g.drawString("Press 'Start' to begin", 845, 590); // Adds the string to just below and
to the left of "EVSIM"
12.    }

```

Java



Despite this working and displaying the content near the centre of the screen, it was inaccurate, and was not correct when displaying across different monitor sizes. For example, on a smaller monitor, or if the window size was changed by the user, the text was not near the centre, or not showing.

To rectify this issue, I changed the x and y position for the title and subtitle to be (current window size - textSize) / 2. This is demonstrated in the code below. After re-testing on multiple monitors, and changing the window size, the menu is now fluid.

```
1.      @Override
2.      protected void paintComponent(Graphics g) {
3.          super.paintComponent(g);
4.          Dimension gameWindowSize = getSize(); // Gets the current window size as a Dimension
5.          FontMetrics fontMetrics = g.getFontMetrics(); // Class to get the characteristics of a
string of text
6.
7.          g.setColor(Color.white); // Sets the colour to white
8.          g.setFont(new Font("Agency FB", Font.BOLD, 108));
9.
10.         int textWidth = fontMetrics.stringWidth("EVSIM"); // Gets the width of the title
"EVSIM"
11.         int textHeight = fontMetrics.getHeight(); // Returns the height of the font (which
would be 108 in this case).
12.
13.         // variable for the x and y position of where the title should be.
14.         int titleTextX = (gameWindowSize.width - textWidth) / 2 - 100; // -100 to account for
the size of the menu on the left
15.         int titleTextY = (gameWindowSize.height - textHeight) / 2;
16.         // Takes the size of the game window, e.g. 1920, takes away the size of the text, e.g.
17.         // 35, and divides by 2 to get the exact centre on any window size.
18.
19.         g.drawString("EVSIM", titleTextX, titleTextY); // Draws the string to the page at
position x and y
20.
21.         g.setFont(new Font("Agency FB", Font.BOLD, 36));
22.         g.drawString("Press 'Start' to begin", titleTextX - 15 /*Left of title*/, titleTextY +
50 /*Below the title*/);
23.         // Draws the subtitle on the page just below and to the left of the title.
24.     }
```

Java



Image above representing the default window size.

Image below representing the window when dimensions are changed.

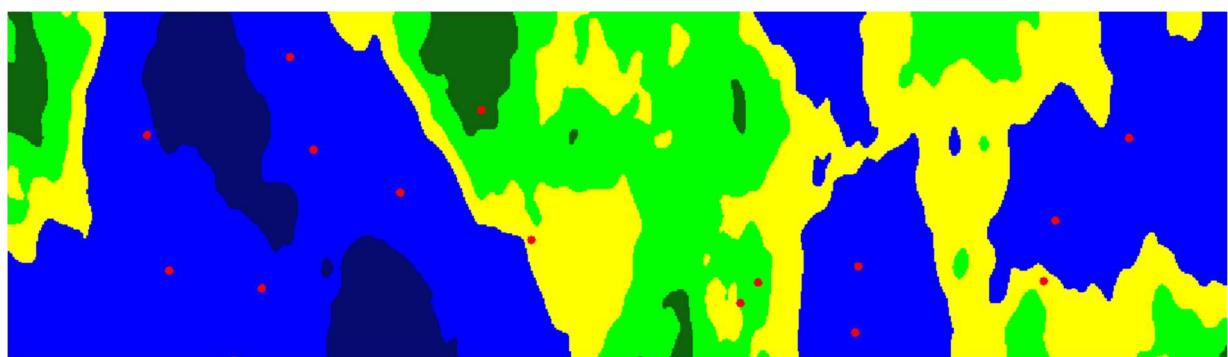


3.1.3 Display of the organisms

Display of the organisms within front end UI is only a very small section, linking back to 2.2, the front-end UI only needs to display the organisms and link it to the backend. This section here will create a user defined number of organisms at a random position, and initialise them into a JSON file as well.

3.1.3.1 Initial creation of organisms

Initialising the organisms into new positions at the start of the simulation is very simple. The program needs to take in the window size, and generate an organism at a random location within that window space. At the moment the organisms are going to be displayed as a red circle with a radius of 3 pixels. This should be visible enough for the user to make them out, as well as click on them. The user should be able to change the organism radius, number of organisms to generate as well as the default organism colour in the main menu settings. The default number of organisms to generate is 50, and should be capped at 750. Therefore this program needs to accept changes from global settings, but also have default settings in the event settings are not changed.



The above image shows what the organisms would look like under default settings. The red contrasts with other colours on the map, therefore, should be able to be seen fine under the default settings.

The code used to randomly generate these organisms is below.

```
1.  public static int WIDTH = 1920; // Default is 1920, can be changed
2.  public static int HEIGHT = 1080; // Default is 1080, can be changed
3.  private static final int ORGANISM_COUNT = 50; // Default is 50, can be changed
4.  private Random random = new Random();
5.  public int x;
6.  public int y;
7.  public int ORGANISM_RADIUS = 3; // Default is 3, can be changed
8.
9.  private void generateOrganisms(Graphics g) {
```

```

10.     g.setColor(Color.RED);
11.     for (int organismCounter = 0; organismCounter < ORGANISM_COUNT; organismCounter++){
12.         newOrganismLocation(); // New values for x and y
13.         //WRITE TO JSON
14.         g.fillOval(x, y, 2 * ORGANISM_RADIUS, 2 * ORGANISM_RADIUS);
15.     }
16. }
17. private void newOrganismLocation(){
18.     x = random.nextInt(WIDTH - 2 * ORGANISM_RADIUS) + ORGANISM_RADIUS; // Random x and y
coordinate on the grid.
19.     y = random.nextInt(HEIGHT - 2 * ORGANISM_RADIUS) + ORGANISM_RADIUS;
20. }

```

Java

3.1.3.2 Creating and writing to the corresponding JSON file

This will reflect what was laid out in 2.3.2.2, with the same default attributes. To read and write to JSON in Java, I have had to import the json.simple library, one commonly used in other programs to read and write to JSON. This required an external JAR file to be added but is now part of the project.

I will be using this library to write to a file called organisms.json, mirroring what was planned in design. This would require the program to create and write an object to the file for each organism. To copy the same format as design for each organism I would need to use two JSON objects, using the class JSONObject. One of these object's would contain all of the information on the organism, with the other embedded inside, containing all of the organisms stats. Once the objects have been created, they will be sent to the JSON file itself, which I will do with the use of FileWriter, a standard Java library. The code to achieve this is presented below, as well as the result:

```

1. import org.json.simple.JSONObject;
2. import java.io.IOException;
3. import java.io.FileWriter;
4. private void generateOrganisms(Graphics g) {
5.     g.setColor(Color.RED);
6.     String filename = "organisms.json"; // the file for the program to write to
7.     JSONObject jsonObject = new JSONObject(); // create a new json object
8.     for (int organismCounter = 0; organismCounter < ORGANISM_COUNT; organismCounter++) {
9.         newOrganismLocation();
10.        // Adds a key/value pair to the json object initialised above
11.        jsonObject.put("organismId", organismCounter);
12.        jsonObject.put("x-coordinate", x);
13.        jsonObject.put("y-coordinate", y);
14.        jsonObject.put("currentTerrain", "getTerrain(x, y)"); // Function yet to be made
15.
16.        JSONObject statsObject = new JSONObject(); // The stats object is initialised here
17.        statsObject.put("currentForm", 0); // Adding each key/value pair to the stats
object
18.        statsObject.put("retentionRate", 50);
19.        statsObject.put("age", 0);
20.        statsObject.put("health", 100);
21.        statsObject.put("speed", 1);
22.        statsObject.put("awareness", 0);
23.        statsObject.put("size", 10);
24.        statsObject.put("attack", 0);
25.        statsObject.put("defence", 0);
26.        statsObject.put("preferredTemperature", null);
27.        statsObject.put("preferredFood", null);
28.
29.        jsonObject.put("stats", statsObject); // This adds the statsObject (created above)
to the key "stats"
30.        jsonObject.put("preferredTerrain", null);
31.        jsonObject.put("isDead", false);
32.
33.        // Write the JSON object to a file
34.        try (FileWriter fileWriter = new FileWriter(filename)) {
35.            fileWriter.write(jsonObject.toJSONString()); // This adds the whole json object
to the file filename (defined above)
36.            System.out.println("JSON object has been written to " + filename); // To check
if the organism has been created

```

```

37.         } catch (IOException e) {
38.             e.printStackTrace(); // To catch any errors
39.         }
40.     }
41.     System.out.println(jsonObject); // To show in the console what has been added
42. }

```

Java

The result from this code was not perfect, but showed that the JSON.simple library would be sufficient at creating a JSON object in this instance. The result is shown below, due to there being checking statements in the code, we can see what has been generated in the JSON file from here.

```

45 JSON object has been written to organisms.json
46 JSON object has been written to organisms.json
47 JSON object has been written to organisms.json
48 JSON object has been written to organisms.json
49 JSON object has been written to organisms.json
50 JSON object has been written to organisms.json
51 JSON object has been written to organisms.json
52 {"organismId":49,"stats": {"currentForm":0,"awareness":0,"size":10,"defence":0,"attack":0
    , "retentionRate":50,"health":100,"preferredFood":null,"age":0,"speed":1,
    "preferredTemperature":null}, "preferredTerrain":null, "currentTerrain": "getTerrain(x, y)", "
    x-coordinate":1581, "y-coordinate":61, "isDead":false}

```

Lines 45 to 51 show that the program is correctly executing the code without fault, however upon inspection of the final JSON file, only the last organism that is generated (see line 52 above), has been saved. This is due to the format of JSON, and the way I have chosen to write to the file. Regarding JSON, for multiple objects to be written to one file, there needs to be an object array. An object array has not been created here, therefore every object overwrites the other.

To rectify this, I remain using json.simple, as it has the class JSONArray as well. The implementation for this is based off the same principles, except the jsonObject will be defined and updated inside the for loop, with it being added to a jsonArray (defined outside) each time. This jsonArray will then be added at the end to the actual file. The code and resultant file is displayed below.

```

1. import org.json.simple.JSONArray;
2. import org.json.simple.JSONObject;
3. import java.io.IOException;
4. import java.io.FileWriter;
5. private void generateOrganisms(Graphics g) {
6.     g.setColor(Color.RED);
7.     String filename = "organisms.json"; // The file for the program to write to
8.     JSONArray jsonArray = new JSONArray(); // Creates a new JSON Array
9.     for (int organismCounter = 0; organismCounter < ORGANISM_COUNT; organismCounter++) {
10.         JSONObject jsonObject = new JSONObject(); // Creates a new JSON object for each
organism
11.         newOrganismLocation(); // generates new x and y coordinates
12.
13.         // Adds a key/value pair to the json object initialised above
14.         jsonObject.put("organismId", organismCounter);
15.         jsonObject.put("x-coordinate", x);
16.         jsonObject.put("y-coordinate", y);
17.         jsonObject.put("currentTerrain", "getTerrain(x, y)"); // Function yet to be made
18.
19.         JSONObject statsObject = new JSONObject(); // The stats object is initialised
20.         statsObject.put("currentForm", 0); // Adding each key/value to the stats
21.         statsObject.put("retentionRate", 50);
22.         statsObject.put("age", 0);
23.         statsObject.put("health", 100);
24.         statsObject.put("speed", 1);
25.         statsObject.put("awareness", 0);
26.         statsObject.put("size", 10);
27.         statsObject.put("attack", 0);
28.         statsObject.put("defence", 0);
29.         statsObject.put("preferredTemperature", null);
30.         statsObject.put("preferredFood", null);

```

```

31.         jsonObject.put("stats", statsObject);
32.         jsonObject.put("preferredTerrain", null);
33.         jsonObject.put("isDead", false);
34.
35.     jsonArray.add(jsonObject); // Adding the individual objects to the array
36.
37. }
38. // Write the JSON array to the file
39. try (FileWriter fileWriter = new FileWriter(filename)) {
40.     fileWriter.write(jsonArray.toJSONString());
41.     fileWriter.flush();
42.     System.out.println("JSON array has been written to " + filename);
43. } catch (IOException e) {
44.     e.printStackTrace();
45. }
46. }
47. System.out.println(jsonArray); // To show in the console what has been added
48.
}

```

Java

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=51552:C:\Program Files\JetBrains
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\
Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar Main
2 JSON array has been written to organisms.json
3 [{"organismId":0,"stats":{"currentForm":0,"awareness":0,"size":10,"defence":0,"attack":0
,"retentionRate":50,"health":100,"preferredFood":null,"age":0,"speed":1,
"preferredTemperature":null},"preferredTerrain":null,"currentTerrain":"getTerrain(x, y)","
x-coordinate":1356,"y-coordinate":455,"isDead":false},{ {"organismId":1,"stats":{"
currentForm":0,"awareness":0,"size":10,"defence":0,"attack":0,"retentionRate":50,"health"
":100,"preferredFood":null,"age":0,"speed":1,"preferredTemperature":null},
"preferredTerrain":null,"currentTerrain":"getTerrain(x, y)","x-coordinate":294,"y-
coordinate":248,"isDead":false},{ {"organismId":2,"stats":{ "currentForm":0,"awareness":0,
"size":10,"defence":0,"attack":0,"retentionRate":50,"health":100,"preferredFood":null,"age"
":0,"speed":1,"preferredTemperature":null}, "preferredTerrain":null,"currentTerrain":"
getTerrain(x, y)","x-coordinate":617,"y-coordinate":419,"isDead":false}]

```

As seen in the output, this method of using an array, works. Organisms.json now contains the information for 3 organisms, which can be written and read to across the whole project.

3.1.3.3 Using the stored information

The original method for reading JSON to Java was to use a function from a JS project. This function returns the JSON file as a JSON object in JavaScript, allowing for it to be manipulated as well as read. Implementing this into Java however was initially not fruitful due to there being no assignable JSON object within the default environment. However, json.simple also includes JSON Object and JSON Array as classes, meaning the JSON could be parsed into a JSON Array (JSON file contains array of objects).

Below is the original JS code, as well as the mirrored Java code in which I attempted to use.

```

1.     function loadJSON(filename = '') {
2.         return JSON.parse(
3.             fs.existsSync(filename)
4.                 ? fs.readFileSync(filename).toString()
5.                 : '')
6.     }
7. };

```

JavaScript

```

1.     public static JSONArray loadJSON(String filename) {
2.         try {
3.             BufferedReader reader = new BufferedReader(new FileReader(filename)); // Reads the
file
4.             StringBuilder jsonString = new StringBuilder(); // String to be parsed
5.             String line;
6.             while ((line = reader.readLine()) != null) { // While not at end of file

```

```

7.             jsonString.append(line); // Adding in each JSON Object
8.         }
9.     reader.close();
10.
11.    JSONParser parser = new JSONParser();
12.    return (JSONArray) parser.parse(jsonString.toString()); // parsing and returning
13. } catch (IOException | ParseException e) {
14.     e.printStackTrace();
15.     return new JSONArray(); // Returns blank JSONArray otherwise
16. }
17. }
```

Java

The function here worked as intended, however I was only able to take one object from the array which was created. Using the code block below, only one object could be retrieved, without the ability to fetch one value, such as the organism ID. This issue with this is that a second function would be required to read this object, in order to return each value inside.

```

1. JSONArray testArray = loadJSON(filename);
2. System.out.println(testArray.get(0));
```

Java

1. "C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\lib\idea_rt.jar=57767:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;
2. {"xCoordinate":358,"organismId":0,"yCoordinate":145,"stats":{"currentForm":0,"awareness":0,"size":10,"defence":0,"attack":0,"retentionRate":50,"health":100,"preferredFood":null,"age":0,"speed":1,"preferredTemperature":null},"preferredTerrain":null,"currentTerrain":"Grass","isDead":false}

The image above represents the output, with only the first object being returned.

This lead to me to look for alternative options. To efficiently read all the values from the JSON file, I would need to use the Jackson library for Java. This allows for JSON data to be parsed into an class, therefore allowing for anything to be read at any one point. I will be specifically using the ObjectMapper class from Jackson, to parse the JSON data into an array. This would require each variable to be parsed into variables within the code. To best meet their requirements, a class Organism and Stats will be created. The Stats class will define the variables from the object "stats", with the Organism class defining all other variables. Due to the fact that each of the variables is declared under the same name as the ones found in the JSON file, the ObjectMapper can set each object to its related variable. The two classes can be viewed in the code block below.

```

1. public class Organism {
2.     private int organismId; // Declaring each variable
3.     private Stats stats;
4.     private String preferredTerrain;
5.     private String currentTerrain;
6.     private int xCoordinate;
7.     private int yCoordinate;
8.     private boolean isDead;
9. }
```

Java

```

1. public class Stats {
2.     private int currentForm; // Declaring each variable
3.     private int awareness;
4.     private int size;
5.     private int defence;
6.     private int attack;
```

```

7.     private int retentionRate;
8.     private int health;
9.     private String preferredFood;
10.    private int age;
11.    private int speed;
12.    private String preferredTemperature;
13. }
```

Java

To access and change these variables elsewhere, get and set methods will also have to be created for each organism. As seen below, with both contained in their representative classes.

Organism:

```

1.  public void setOrganismId(int organismId) { // Getter and setter methods for each variable
2.      this.organismId = organismId;
3.  }
4.  public Stats getStats() {
5.      return stats;
6.  }
7.  public void setStats(Stats stats) {
8.      this.stats = stats;
9.  }
10. public String getPreferredTerrain() {
11.     return preferredTerrain;
12. }
13. public void setPreferredTerrain(String preferredTerrain) {
14.     this.preferredTerrain = preferredTerrain;
15. }
16. public String getCurrentTerrain() {
17.     return currentTerrain;
18. }
19. public void setCurrentTerrain(String currentTerrain) {
20.     this.currentTerrain = currentTerrain;
21. }
22. public int getxCoordinate() {
23.     return xCoordinate;
24. }
25. public void setxCoordinate(int xCoordinate) {
26.     this.xCoordinate = xCoordinate;
27. }
28. public int getyCoordinate() {
29.     return yCoordinate;
30. }
31. public void setyCoordinate(int yCoordinate) {
32.     this.yCoordinate = yCoordinate;
33. }
34. public boolean isIsDead() {
35.     return isDead;
36. }
37. public void setIsDead(boolean isDead) {
38.     this.isDead = isDead;
39. }
```

Java

Stats:

```

1.  public int getCurrentForm() { // Getter and setter methods for each variable
2.      return currentForm;
3.  }
4.  public void setCurrentForm(int currentForm) {
5.      this.currentForm = currentForm;
6.  }
7.  public int getAwareness() {
8.      return awareness;
9.  }
10. public void setAwareness(int awareness) {
11.     this.awareness = awareness;
12. }
13. public int getSize() {
14.     return size;
```

```

15.    }
16.    public void setSize(int size) {
17.        this.size = size;
18.    }
19.    public int getDefence() {
20.        return defence;
21.    }
22.    public void setDefence(int defence) {
23.        this.defence = defence;
24.    }
25.    public int getAttack() {
26.        return attack;
27.    }
28.    public void setAttack(int attack) {
29.        this.attack = attack;
30.    }
31.    public int getRetentionRate() {
32.        return retentionRate;
33.    }
34.    public void setRetentionRate(int retentionRate) {
35.        this.retentionRate = retentionRate;
36.    }
37.    public int getHealth() {
38.        return health;
39.    }
40.    public void setHealth(int health) {
41.        this.health = health;
42.    }
43.    public String getPreferredFood() {
44.        return preferredFood;
45.    }
46.    public void setPreferredFood(String preferredFood) {
47.        this.preferredFood = preferredFood;
48.    }
49.    public int getAge() {
50.        return age;
51.    }
52.    public void setAge(int age) {
53.        this.age = age;
54.    }
55.    public int getSpeed() {
56.        return speed;
57.    }
58.    public void setSpeed(int speed) {
59.        this.speed = speed;
60.    }
61.    public String getPreferredTemperature() {
62.        return preferredTemperature;
63.    }
64.    public void setPreferredTemperature(String preferredTemperature) {
65.        this.preferredTemperature = preferredTemperature;
66.    }

```

Java

With the classes now set up, the object mapper can be called. For the display of organisms, I will be only needing to paint the organisms at their representative x and y coordinates, therefore each organism (in the JSON file) must be iterated through and painted. Within the paintComponent method of generateOrganisms.java, I will define an array of the organisms class, with each object in the JSON file being an element of the array. The array will then be iterated through, with the x and y coordinates at each index being used to paint the organism to the screen.

```

1.    public void paintComponent(Graphics g) {
2.        super.paintComponent(g);
3.        ObjectMapper objectMapper = new ObjectMapper(); // Initialising the object mapper
4.        try {
5.            Organism[] organisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
// objectMapper.readValue(a,b) takes the data from the file (a), and parses it into the array (b)
6.            for (Organism organism : organisms) { // For each organism in organisms

```

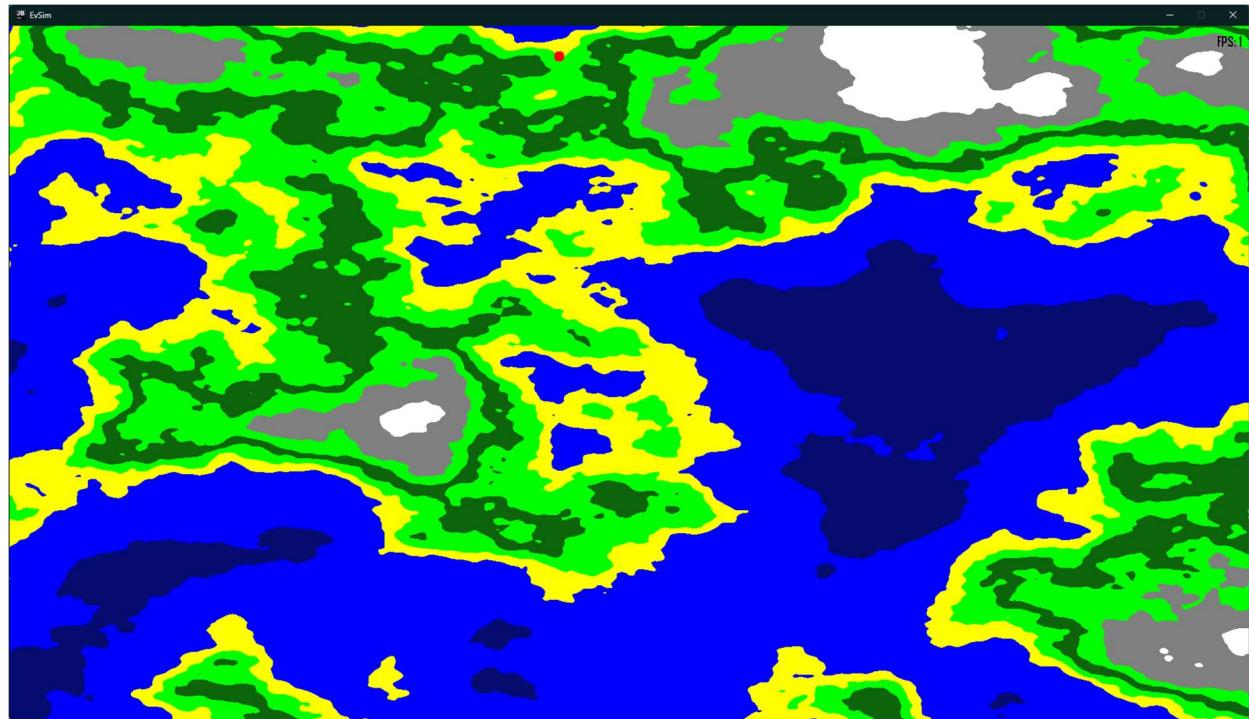
```

7.                 g.setColor(Color.RED);
8.                 g.fillOval(organism.getxCoordinate(), organism.getyCoordinate(), 5 *
ORGANISM_RADIUS, 5 * ORGANISM_RADIUS); // Paint a red oval at position x, y, dimension 5,5
9.             }
10.        } catch (IOException e) {
11.            throw new RuntimeException(e); // Any errors thrown
12.        }
13.    }

```

Java

Testing this solution, with anything from 1 to 500 organisms works as intended, with the organisms being painted at their respective coordinates.



Red dot demonstrating the organism, size exaggerated for the purpose of this image. Below is the respective JSON format for this organism which is contained in organisms.json.

```

1.[{"xCoordinate":785,"organismId":0,"yCoordinate":36,"stats":{"currentForm":0,"awareness":0,"size":10,"defence":0,"attack":0,"retentionRate":50,"health":100,"preferredFood":null,"age":0,"speed":1,"preferredTemperature":null,"preferredTerrain":null,"currentTerrain":"Beach","isDead":false}]

```

JSON

3.1.3.4 Final code

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.io.File;
4. import java.util.Random;
5. import org.json.simple.JSONArray;
6. import org.json.simple.JSONObject;
7. import java.io.IOException;
8. import java.io.FileWriter;
9. import com.fasterxml.jackson.databind.ObjectMapper;
10.
11. public class generateOrganisms extends JPanel {
12.     private static int ORGANISM_COUNT = 1; // Default is 50, can be changed
13.     private Random random = new Random();
14.     private int x;
15.     private int y;
16.     public int ORGANISM_RADIUS = 3; // Default is 1, can be changed
17.     public generateOrganisms(int width, int height) {
18.         String filename = "organisms.json"; // The file for the program to write to
19.         JSONArray jsonArray = new JSONArray(); // Creates a new JSON Array
20.         for (int organismCounter = 0; organismCounter < ORGANISM_COUNT; organismCounter++) {

```

```

21.         JSONObject jsonObject = new JSONObject(); // Creates a new JSON object for each
22. organism
23.         newOrganismLocation(width, height); // generates new x and y coordinates
24.
25.         // Adds a key/value pair to the json object initialised above
26.         jsonObject.put("organismId", organismCounter);
27.         jsonObject.put("xCoordinate", x);
28.         jsonObject.put("yCoordinate", y);
29.         jsonObject.put("currentTerrain", generateMap.getTerrain(x, y));
30.
31.         JSONObject statsObject = new JSONObject(); // The stats object is initialised
32. statsObject.put("currentForm", 0); // Adding each key/value to the stats
33. statsObject.put("retentionRate", 50);
34. statsObject.put("age", 0);
35. statsObject.put("health", 100);
36. statsObject.put("speed", 1);
37. statsObject.put("awareness", 0);
38. statsObject.put("size", 10);
39. statsObject.put("attack", 0);
40. statsObject.put("defence", 0);
41. statsObject.put("preferredTemperature", null);
42. statsObject.put("preferredFood", null);
43.
44.         jsonObject.put("stats", statsObject);
45.         jsonObject.put("preferredTerrain", null);
46.         jsonObject.put("isDead", false);
47.
48.         jsonArray.add(jsonObject); // Adding the individual objects to the array
49.
50.     }
51.     // Write the JSON array to the file
52.     try (FileWriter fileWriter = new FileWriter(filename)) {
53.         fileWriter.write(jsonArray.toJSONString());
54.         fileWriter.flush();
55.         //System.out.println("JSON array has been written to " + filename);
56.     } catch (IOException e) {
57.         System.out.println(e);
58.     }
59.     System.out.println(jsonArray); // Testing to see what has been added to the file
60. }
61. private void newOrganismLocation(int Width, int Height){ // Generates a random x and y
coordinate
62.     x = random.nextInt(0, Width);
63.     y = random.nextInt(0, Height);
64. }
65. public void paintComponent(Graphics g) {
66.     super.paintComponent(g);
67.     ObjectMapper objectMapper = new ObjectMapper();
68.     try {
69.         Organism[] organisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
70.         // objectMapper.readValue(a,b) takes the data from the file (a), and parses it into
the array (b)
71.         for (Organism organism : organisms) { // For each organism (value) in Organisms
(array)
72.             g.setColor(Color.RED);
73.             g.fillOval(organism.getxCoordinate(), organism.getyCoordinate(), 5 *
ORGANISM_RADIUS, 5 * ORGANISM_RADIUS); // Paint a red oval at position x, y, dimension 5, 5
74.             // Can be changed at any point, to be an image, or a different shape
75.         }
76.     } catch (IOException e) {
77.         throw new RuntimeException(e);
78.     }
79. }

```

Java

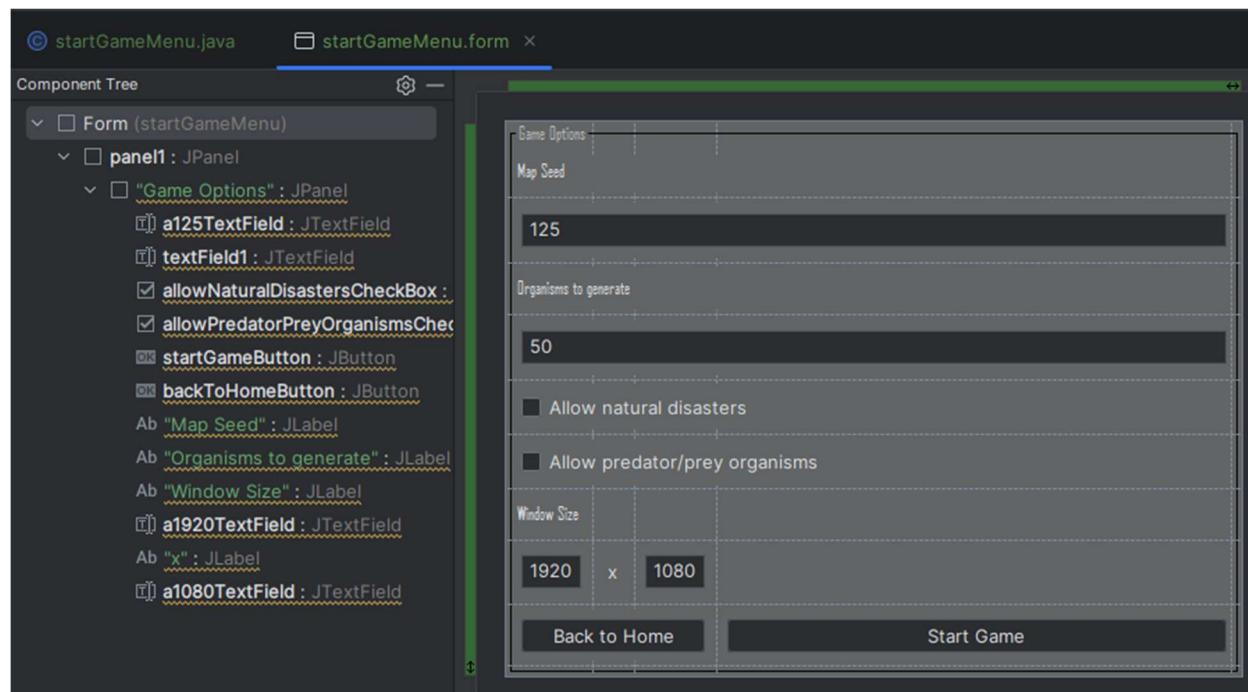
Keeping in mind that this is purely front-end UI at this stage, meaning the produced JSON file variables have the potential to change down the line, with factors added or removed, as will happen with the Organism and Stats classes.

3.1.4 Start game options menu

This section will focus on the settings that the user can define before the iteration begins. Settings that the user can define before the game starts are those set during map and organism generation, for example organism count, and map seed. The user will also have the opportunity to change the generating window size before they start the game as well, in case they haven't set a default in the main game options menu (3.1.5). For later development, I will also allow the user to change whether natural disasters and predator/prey cycles can spawn within this menu.

3.1.4.1 Start Game Menu Design

For this, I will use IntelliJ's inbuilt form creator, which allows for customisation through a drag and drop to create a form. This makes it easier to style the UI without having to iteratively test UI. As aforementioned, this menu will only require very few input parameters, therefore this menu will include 3 text input fields, and 2 checkboxes. Thanks to how forms within IntelliJ works, all the formatting code is done for you, therefore only the panel itself has to be added for it to show. This is what the form design looks like within IntelliJ, followed by the code to add it into the window upon the click of 'Start'.



```
1. import javax.swing.*;
2. import java.awt.*;
3.
4. public class startGameMenu {
5.     public JPanel panel1;
6.     JFrame frame;
7.     public startGameMenu() {
8.         panel1.setPreferredSize(new Dimension(500,500)); // Both the panel and the frame same size.
9.         frame = new JFrame("Start Game");
10.        frame.setPreferredSize(new Dimension(500,500));
11.        frame.setLocationRelativeTo(null);
12.        frame.add(panel1); // As the panel already contains the text, labels and buttons, only the panel has to be added to the frame.
13.        frame.setResizable(false);
14.        frame.pack();
15.        frame.setVisible(true);
16.    }
17. }
```

Java

Within OpeningMenu.java

```
1. startGameMenu startMenu = new startGameMenu(); // adds the frame when 'Start' is clicked.
```

Java

To finish this off, the program needs to be able to take what the user inputs in each field, and set it in the appropriate class. For example the organisms to generate needs to be sent to generateOrganisms.java, and the seed needs to be sent to generateMap.java

3.1.4.2 Adding functionality to the start game menu

I will go through this in line with the order on the menu, starting with the map seed, and ending with the start game button. For each text box, I will need to set the text to the current setting and take the user input if it is changed. This means that the placeholder text will have to take the current seed from perlinNoise.java, and the organism count from generateOrganisms.java.

Due to the accessibility of IntelliJ forms, taking information from the form itself can be achieved through get methods. The textbox itself is declared as a global JTextField, similar to how panel1 is used.

```
private JTextField seedTextField;
```

Using this, seedTextField.getText() can be used to retrieve what is held in the textbox. In this instance, the value held inside only needs to be fetched and used when 'Start Game' is pressed. The seed that the user inputs will have to be checked, as it needs to be a double, and also needs to be within a realistic range. This will require some testing, and some error checking. At this stage, assuming the seed is in the correct format, it needs to be parsed into a double, and sent into perlinNoise.java. The perlin noise class already contains get and set methods, meaning the random seed can be fetched and shown to the user, and if they change it, it then replaces what is in the class. This is shown below

```
seedTextField.setText(Double.toString(new Random().nextGaussian() * 255));  
// Passes the new seed into a string to be represented in the text field  
perlinNoise.setSeed(Double.parseDouble(seedTextField.getText()));  
// Sets the seed as whatever is in the text box. Input checks will be added before this takes place
```

The function within perlinNoise.java will now also need adapting. The code before generates the seed once it is called, however due to the seed already being generated and set in that class, the constructor only needs to call the class init().

```
1.     public perlinNoise() {  
2.         System.out.println(seed); // Check it is the same as what was sent  
3.         init(); // initialises perlin noise generation  
4.     }
```

Java

Map Seed

-310.3086687032672

The text box now shows the new seed, which the user can change to their needs. Due to the check in line 2 above, it is possible to test the passing of the seed between the two. This was the output from the console.

```
1 "C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\lib\idea_rt.jar=62741:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main  
2 -310.3086687032672
```

As seen, the seed that is generated is identical to the one shown on the menu, meaning it works. The inputs now need to be validated, to check that the input can actually be used. For example if the user doesn't enter a number, then it should not be accepted, giving the user incentive to change their input.

The code below attempts to validate the input from the user. The try catch statement attempts to pass what was held in seedTextField to a double, if it cannot, then it generates an error window, with the error message shown in the string. If the input can be parsed into a double, then it checks if it is within the bounds of -500, and 500. If it is outside of that, then it may cause issues for map generation, therefore the error window asks the user to provide a number within the range. This will be tested along with the rest of the menu within 3.1.4.3.

```

1.     String textInput = seedTextField.getText().trim(); // gets the data from
seedTextField, and removes any blank space (.trim())
2.     try {
3.         double seedInput = Double.parseDouble(textInput); // Parses the trimmed data
from seedTextField into a double
4.         if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
5.             perlinNoise.setSeed(seedInput); // Sets the seed in perlin noise
6.             frame.dispose(); // Removes the start menu
7.         } else {
8.             ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number that is within the range of -1000 and 1000</p></html>"); // If it is not in the bounds, then it displays the ErrorWindow (see below), with html styled text
9.         }
10.        } catch (NumberFormatException error) {
11.            ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number for the seed.<br>This number has to be within the range of
-1000 and 1000</p></html>"); // If there are any errors, it gives the user this error message in
html format
12.        }

```

Java/HTML

The code found for ErrorWindow is below. ErrorWindow can be used across the project, as it takes any string input, which can then be displayed.

```

1. import javax.swing.*;
2. import java.awt.*;
3.
4. public class ErrorWindow {
5.     JFrame frame;
6.     public ErrorWindow(String errorMessage){ // Takes the desired error message as a string
7.         frame = new JFrame("Error");
8.         frame.setPreferredSize(new Dimension(300,150));
9.         JLabel errorLabel = new JLabel(); // Creates a new JLabel which displays the string
10.        errorLabel.setText(errorMessage); // Adds the error message to the created JLabel
11.        frame.add(errorLabel, BorderLayout.CENTER); // Adds this to the centre of the frame
12.        frame.pack();
13.        frame.setResizable(false);
14.        frame.setLocationRelativeTo(null);
15.        frame.setVisible(true);
16.    }
17. }

```

Java

The same now needs to be created for each of the inputs that are asked for in the menu. The full code for all of this will be added below. Refer to comments (//) within the code box for information/justification.

```

1. String textInput = seedTextField.getText().trim(); // Gets the data from seedTextField, and
removes any blank space(.trim())
2.     String text2Input = organismTextField.getText().trim(); // Takes the number of
organisms and removes whitespace (.trim())
3.     String windowHeight = a1920TextField.getText().trim();
4.     String windowWidth = a1080TextField.getText().trim();

```

```

5.         boolean invalidInput = false; // So the game doesn't start with the invalid
information
6.         try {
7.             //MAP SEED
8.             double seedInput = Double.parseDouble(textInput); // Parses the trimmed data
from seedTextField into a double
9.             if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
10.                 perlinNoise.setSeed(seedInput); // Sets the seed in perlin noise
11.             } else {
12.                 // If it is not in the bounds, then it displays the ErrorWindow, with html
styled text
13.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number that is within the range of -1000 and 1000</p></html>");
14.                 invalidInput = true;
15.             }
16.
17.             //ORGANISM COUNT
18.             int organismInput = Integer.parseInt(text2Input); // Passes it into an integer
(cannot accept boolean)
19.             if (organismInput <= 300 && organismInput > 1) { // Bounds
20.                 generateOrganisms.ORGANISM_COUNT = organismInput; // Sets the organism
count to be generated
21.             } else {
22.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number that is within the range of 2 and 50</p></html>");
23.                 invalidInput = true;
24.             }
25.
26.             //WINDOW SIZE
27.             int windowHeightInput = Integer.parseInt(windowHeight); // Passing into Integer
28.             int windowWidthInput = Integer.parseInt(windowWidth);
29.             if(windowHeightInput <= 1080 && windowHeightInput > 200) { // Bounds
30.                 windowHeight = Integer.toString(windowHeightInput); // Sets the window to a
validated Integer
31.             } else {
32.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a window height between 200 and 1080</p></html>");
33.                 invalidInput = true;
34.             }
35.             if(windowWidthInput <= 1920 && windowWidthInput > 200) { // Bounds
36.                 windowWidth = Integer.toString(windowWidthInput); // Sets the window to a
validated Integer
37.             } else {
38.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a window width between 200 and 1920</p></html>");
39.                 invalidInput = true;
40.             }
41.         } catch (NumberFormatException error) {
42.             // If there are any other errors, it gives the user the error message in html
format
43.             ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide valid inputs.<br>One of your inputs contains an unexpected
character</p></html>");
44.             invalidInput = true;
45.         }
46.
47.         if(!invalidInput){
48.             GameWindow gameWindow = new GameWindow(Integer.parseInt(windowWidth),
Integer.parseInt(windowHeight));
49.             gameWindow.startGameThread(); // Creates a new game window from the game window
class, width and height as the input from the user
50.             frame.dispose(); // Removes the start game menu
51.         }

```

Java/HTML

Referring to the comments within this code block, each text field input is validated, and then passed into the appropriate places. Due to the code for predator/prey and natural disasters being implemented much later in design, the checkboxes for these inputs will remain without functionality. Therefore all that is left to add functionality to is the start game button, and the back to home button. To add functionality to these, I will implement MouseListener, a class that allows the program to listen and

react to when the mouse is clicked. To check for which button is pressed, each time the mouse is pressed, a variable (e) is initialised. This can be checked to see if a button is pressed, through .getSource(). Therefore through the line , the button can be listened for, and then carry out the necessary validation. With this added, see below for the full code block of startGameMenu.java. The validation will be tested in 3.1.4.3.

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.MouseEvent;
4. import java.awt.event.MouseListener;
5. import java.util.Random;
6. public class startGameMenu implements MouseListener {
7.     public JPanel panel1;
8.     private JTextField seedTextField;
9.     private JTextField organismTextField;
10.    private JCheckBox allowNaturalDisastersCheckBox; // Not used yet
11.    private JCheckBox allowPredatorPreyOrganismsCheckBox; // Not used yet
12.    private JButton startGameButton;
13.    private JButton backToHomeButton;
14.    private JTextField a1920TextField;
15.    private JTextField a1080TextField;
16.    JFrame frame;
17.    public startGameMenu() {
18.        seedTextField.setText(Double.toString(new Random().nextGaussian() * 255));
19.        perlinNoise.setSeed(Double.parseDouble(seedTextField.getText()));
20.        panel1.setPreferredSize(new Dimension(500,500)); // Both the panel and the frame same
size.
21.        startGameButton.addMouseListener(this);
22.        backToHomeButton.addMouseListener(this);
23.
24.        seedTextField.setText(Double.toString(new Random().nextGaussian() * 255));
25.
26.        frame = new JFrame("Start Game");
27.        frame.setPreferredSize(new Dimension(500,500));
28.        frame.add(panel1); // As the panel already contains the text, labels and buttons, only
the panel has to be added to the frame.
29.        frame.setResizable(false);
30.        frame.pack();
31.        frame.setLocationRelativeTo(null);
32.        frame.setVisible(true);
33.    }
34.
35.    @Override
36.    public void mouseClicked(MouseEvent e) {
37.        if(e.getSource() == startGameButton){ // When they press on 'Start Game', the
information should be validated.
38.            String textInput = seedTextField.getText().trim(); // Gets the data from
seedTextField, and removes any blank space(.trim())
39.            String text2Input = organismTextField.getText().trim(); // Takes the number of
organisms and removes whitespace (.trim())
40.            String windowHeight = a1920TextField.getText().trim();
41.            String windowWidth = a1080TextField.getText().trim();
42.            boolean invalidInput = false; // So the game doesn't start with the invalid
information
43.            try {
44.                //MAP SEED
45.                double seedInput = Double.parseDouble(textInput); // Parses the trimmed data
from seedTextField into a double
46.                if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
47.                    perlinNoise.setSeed(seedInput); // Sets the seed in perlin noise
48.                } else {
49.                    // If it is not in the bounds, then it displays the ErrorWindow, with html
styled text
50.                    ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number that is within the range of -1000 and 1000</p></html>");
51.                    invalidInput = true;
52.                }
53.
54.                //ORGANISM COUNT

```

```

55.             int organismInput = Integer.parseInt(text2Input); // Passes it into an integer
(cannot accept boolean)
56.             if (organismInput <= 300 && organismInput > 1) { // Bounds
57.                 generateOrganisms.ORGANISM_COUNT = organismInput; // Sets the organism
count to be generated
58.             } else {
59.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a number that is within the range of 2 and 50</p></html>");
60.                 invalidInput = true;
61.             }
62.
63.             //WINDOW SIZE
64.             int windowHeightInput = Integer.parseInt(windowHeight); // Passing into
Integer
65.             int windowWidthInput = Integer.parseInt(windowWidth);
66.             if(windowHeightInput <= 1080 && windowHeightInput >= 200) { // Bounds
67.                 windowHeight = Integer.toString(windowHeightInput); // Sets the window to
a validated Integer
68.             } else {
69.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a window height between 200 and 1080</p></html>");
70.                 invalidInput = true;
71.             }
72.             if(windowWidthInput <= 1920 && windowWidthInput >= 200) { // Bounds
73.                 windowWidth = Integer.toString(windowWidthInput); // Sets the window to a
validated Integer
74.             } else {
75.                 ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide a window width between 200 and 1920</p></html>");
76.                 invalidInput = true;
77.             }
78.         } catch (NumberFormatException error) {
79.             // If there are any other errors, it gives the user the error message in html
format
80.             ErrorWindow newError = new ErrorWindow("<html><p style=\"text-
align:center;\">Please provide valid inputs.<br>One of your inputs contains an unexpected
character</p></html>");
81.             invalidInput = true;
82.         }
83.
84.         if(!invalidInput){
85.             GameWindow gameWindow = new GameWindow(Integer.parseInt(windowWidth),
Integer.parseInt(windowHeight));
86.             gameWindow.startGameThread(); // Creates a new game window from the game
window class, width and height as the input from the user
87.             frame.dispose(); // Removes the start game menu
88.         }
89.     } else if(e.getSource() == backToHomeButton){ // If the user desires to go back to
home, display the start menu
90.         OpeningMenu window = new OpeningMenu(1920, 1080);
91.         window.startMenuThread();
92.         frame.dispose(); /// Remove the start menu
93.     }
94. }
95.
96. @Override
97. public void mousePressed(MouseEvent e) {} // Not needed
98.
99. @Override
100. public void mouseReleased(MouseEvent e) {} // Not needed
101.
102. @Override
103. public void mouseEntered(MouseEvent e) {} /// Not needed
104.
105. @Override
106. public void mouseExited(MouseEvent e) {} // Not needed
107. }

```

Java

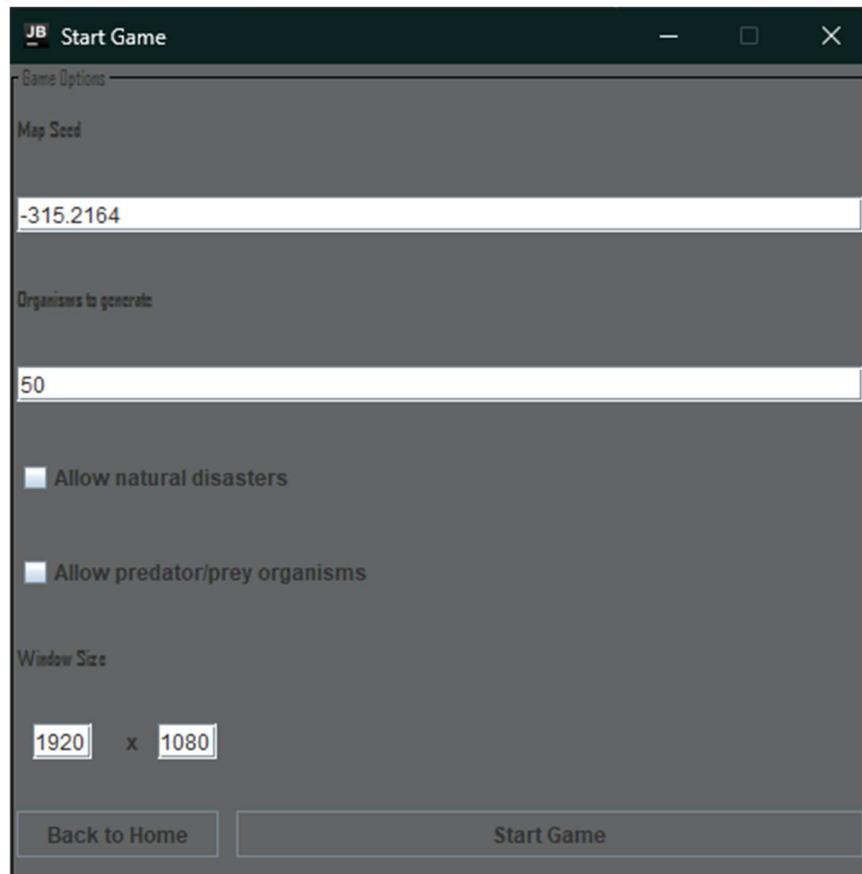
The code behind the different fields, buttons and text boxes remains the same as in 3.1.4.1, with the styling and code defined in startGameMenu.form.

3.1.4.3 Testing start game menu validation

Again, each feature will be tested in linearity, meaning I will start with the seed, and end with the game window size. The table below shows boundary, expected and erroneous data for each text box.

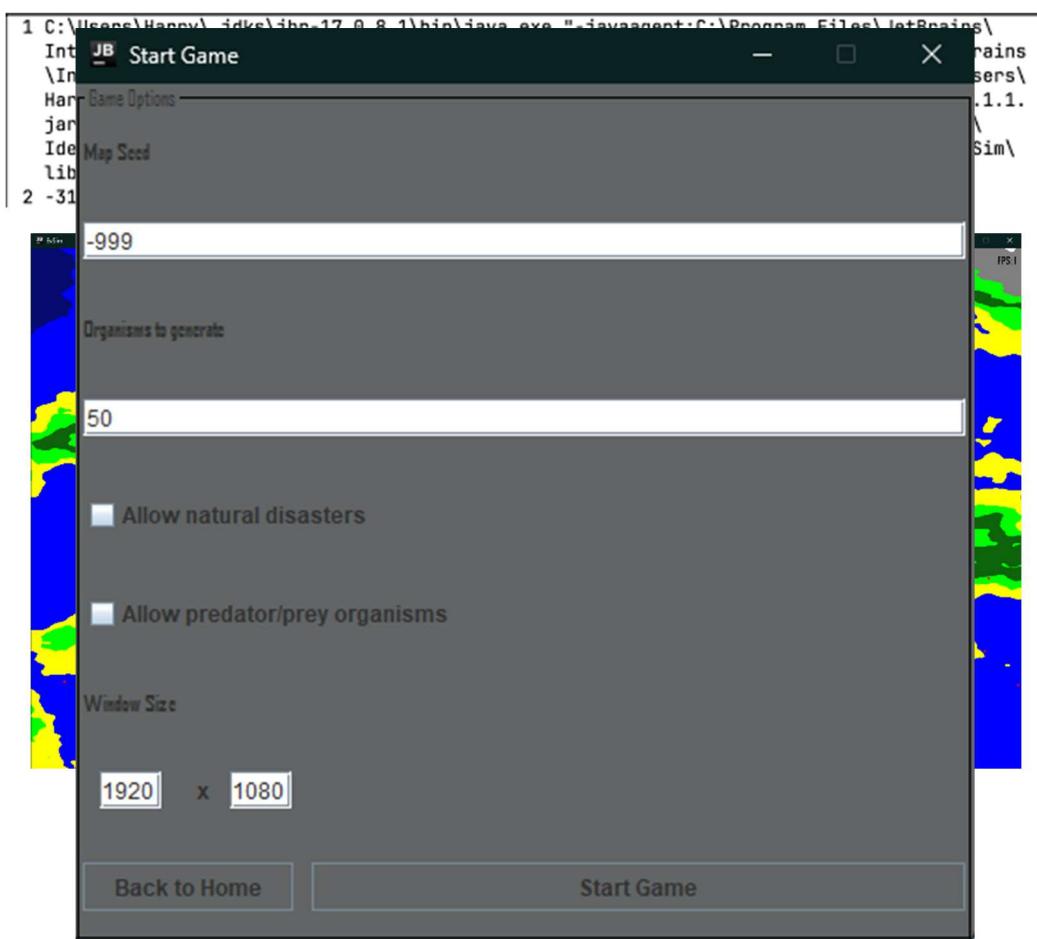
Test data for startGameMenu.java				
Data	Expected	Boundary	Erroneous	Test Result
seedTextField	-315.2164, 65	-999, 999	seed, 3a15, 1025	See 3.1.4.3.1
organismTextField	25, 250	2, 300	-2, organisms, 15a	See 3.1.4.3.2
a1920TextField	300, 1500	200, 1920	width, 4k, 1980	See 3.1.4.3.3
a1080TextField	300, 900	200, 1080	1090, height, 4k	See 3.1.4.3.4

3.1.4.3.1 Testing seedTextField



With the input of -315.2164 in Map Seed, the program should accept this, and load into the game with the seed -315.2164. To ensure it is this seed being generated, the console will send the current seed once the map has loaded.

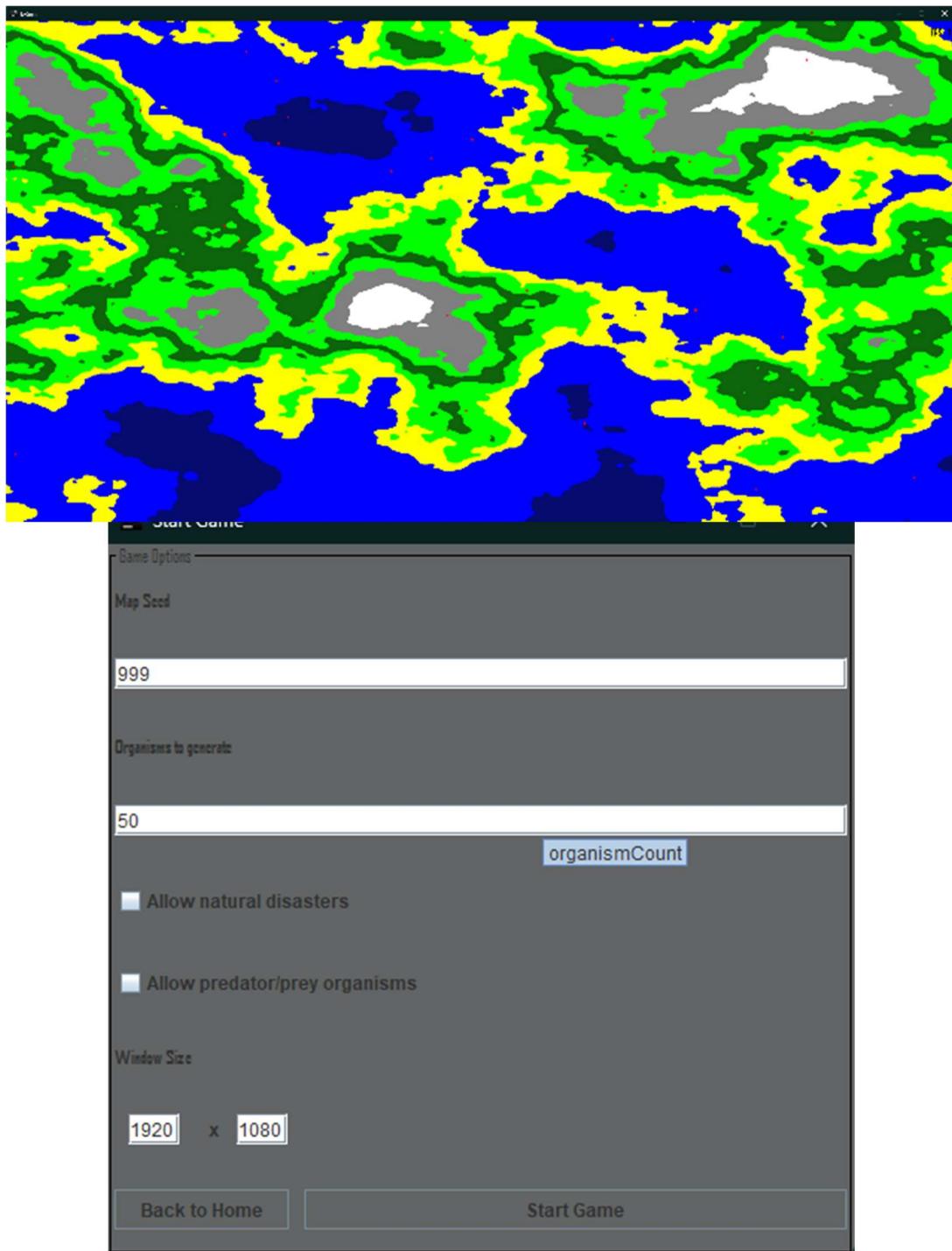
As shown below, the program accepts this as a valid input, and generates the map with the seed - 315.2164.



With the input of 999 and -999, the program should accept this, and generate with the relative seed. Once again images of both the console, map, and start menu will be shown.

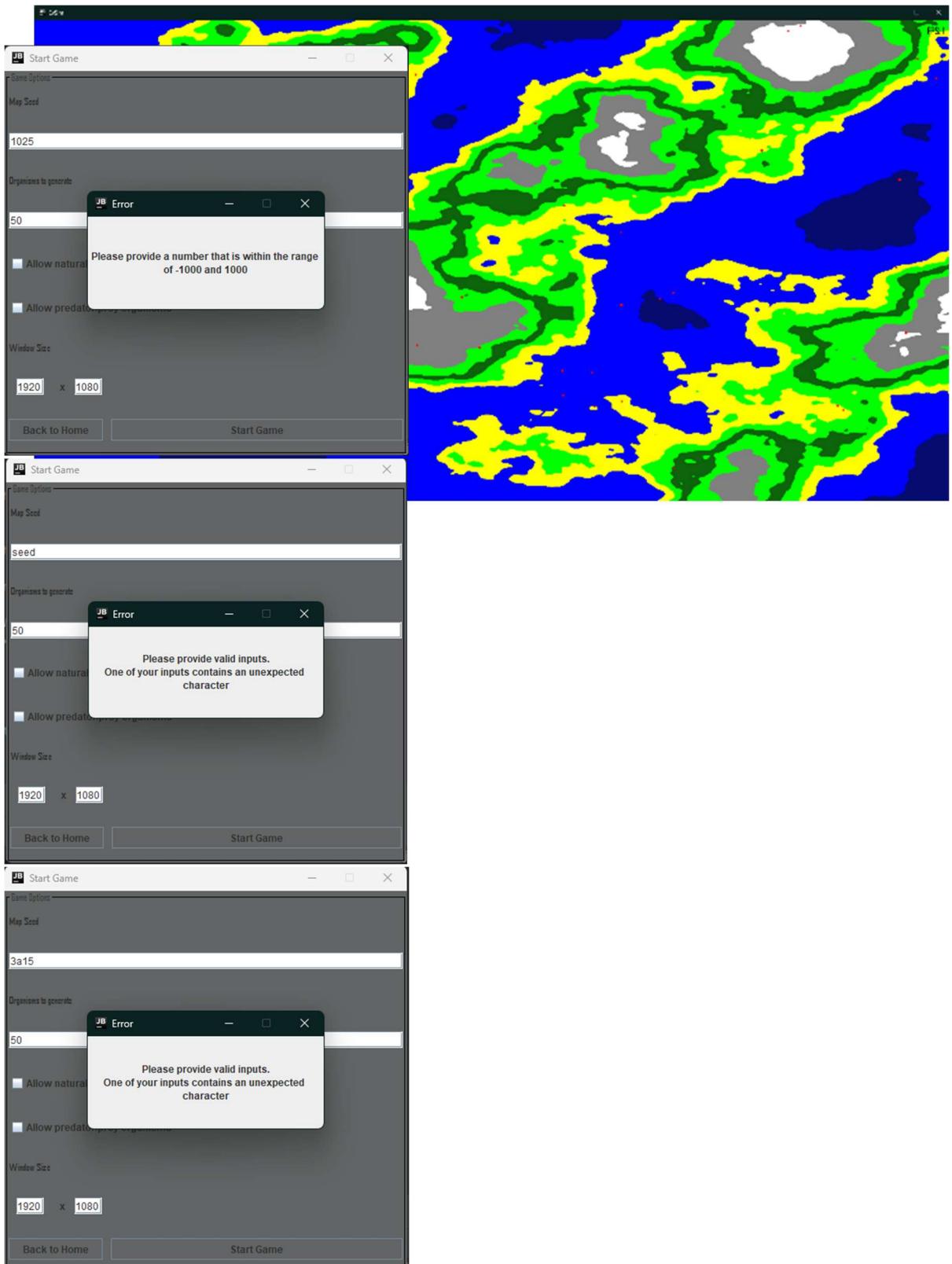
```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=54297:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 -999.0
```

As shown, the boundary data works as expected.



As shown, the boundary data here also works as expected.

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=54279:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 999.0
```

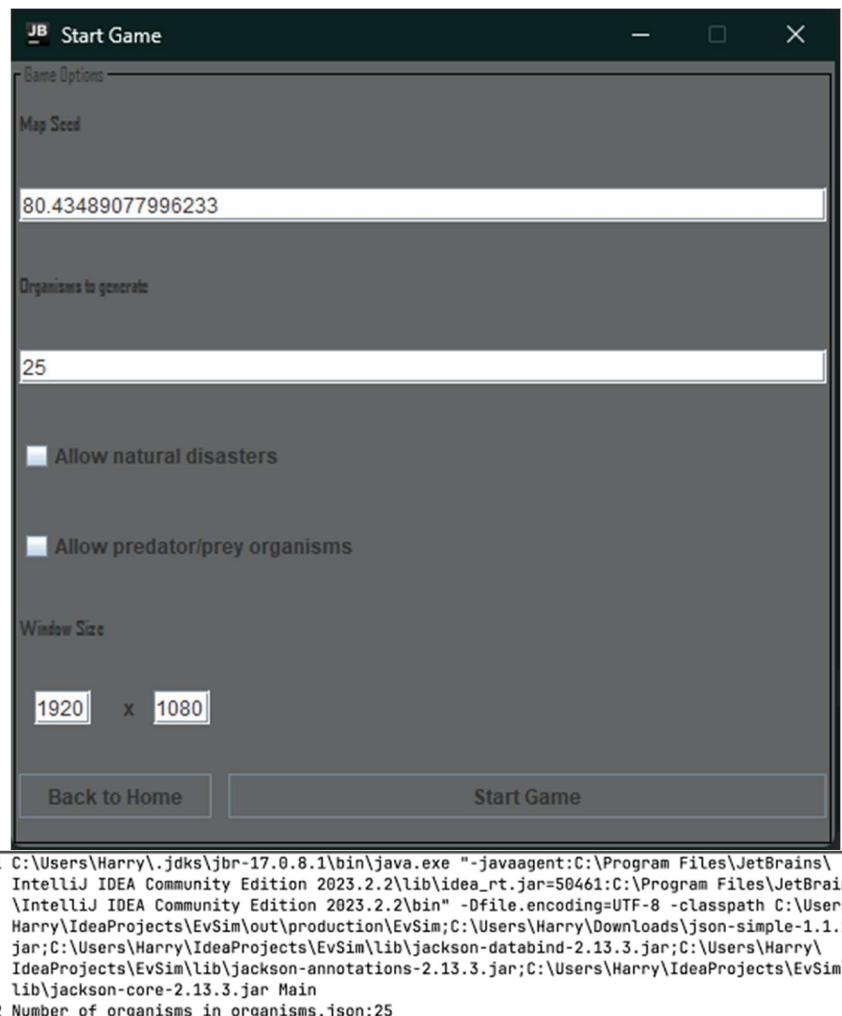


Below is the result of the program with the erroneous data as an input. The program should reject the data and supply the user with an indicative message telling them to check their inputs.

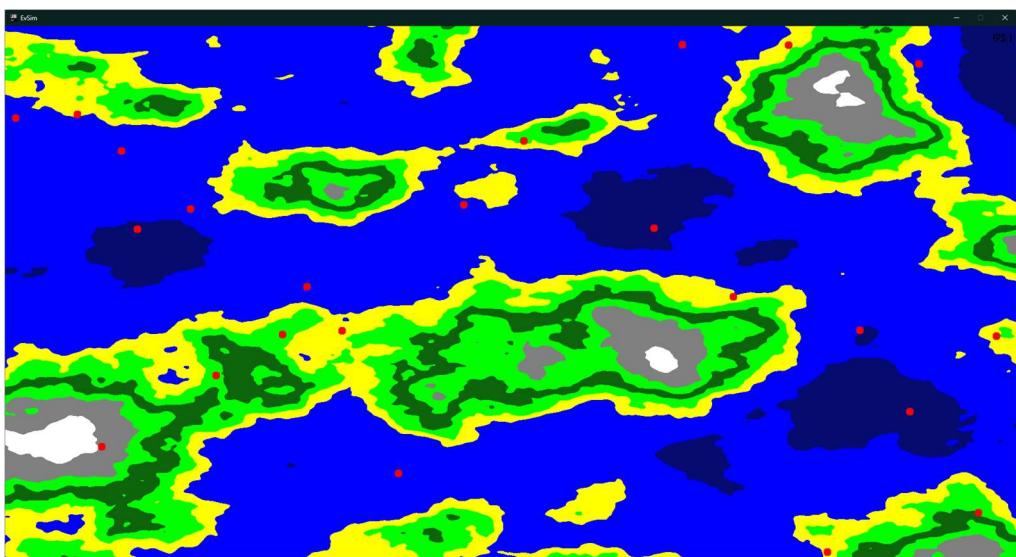
As expected, the program rejects the erroneous data, and tells the user that the input data is wrong. In the case of the number being out of bounds, the program tells the user to enter a value between -1000 and 1000.

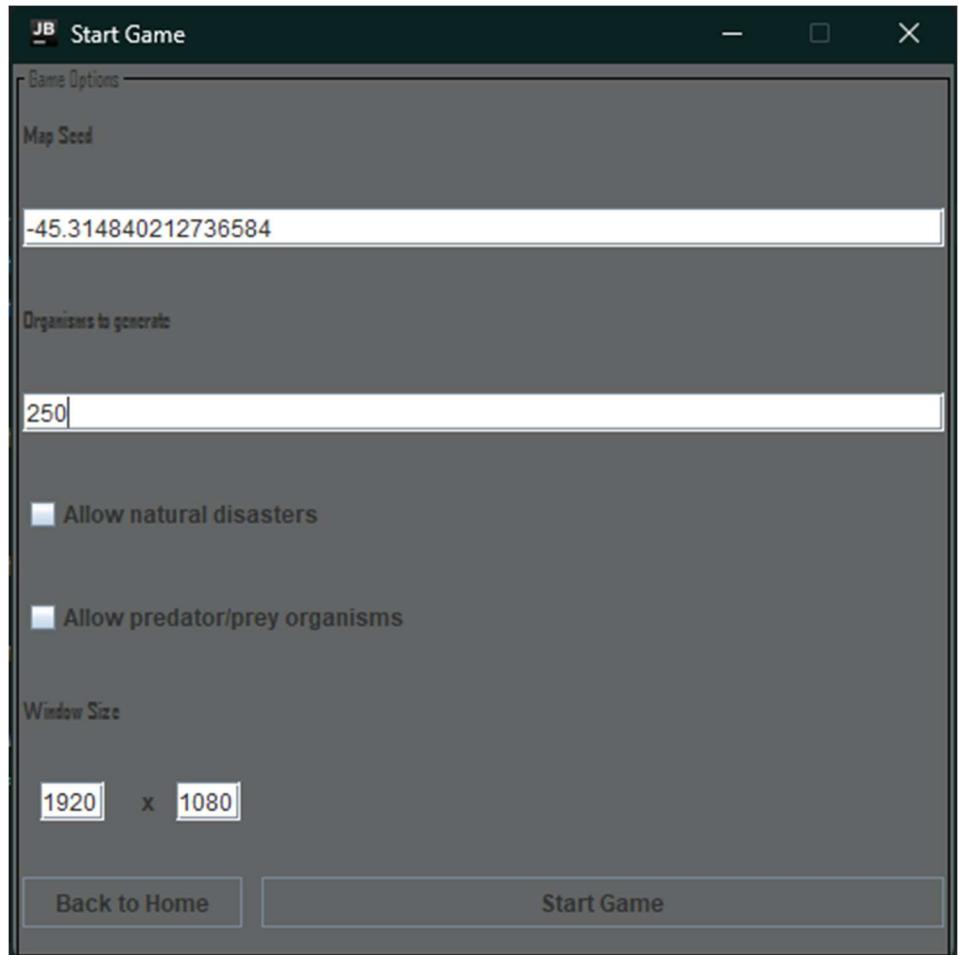
3.1.4.3.2 Testing organismTextField

Starting with the expected data, the program should accept this, and display the corresponding amount of organisms. To test this, I will allow the console to output the total number of organisms generated in the organisms.json file, as well as provide screenshots of both the input, and the game window once it has loaded. The organisms at this current stage are represented by red circles. For this example, the size of the organisms has been emphasised. The focus of this section, is the second text area, labelled "organisms to generate".



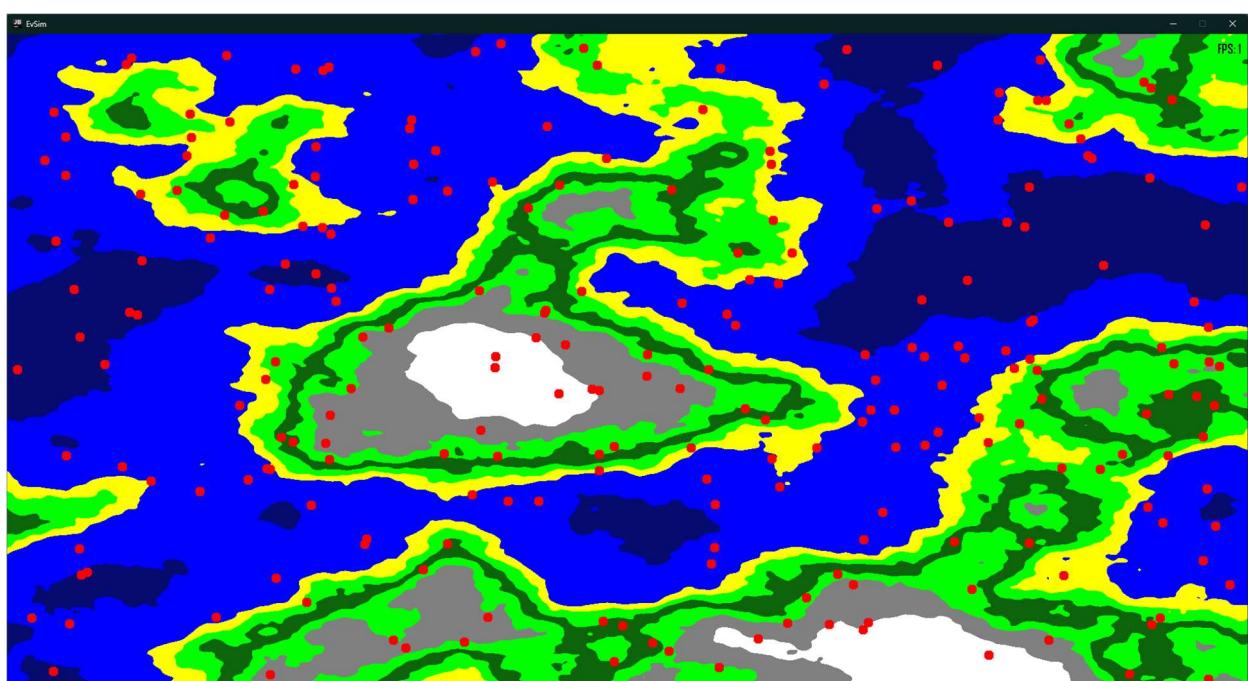
With the expected data entered into the menu, the program works as expected.



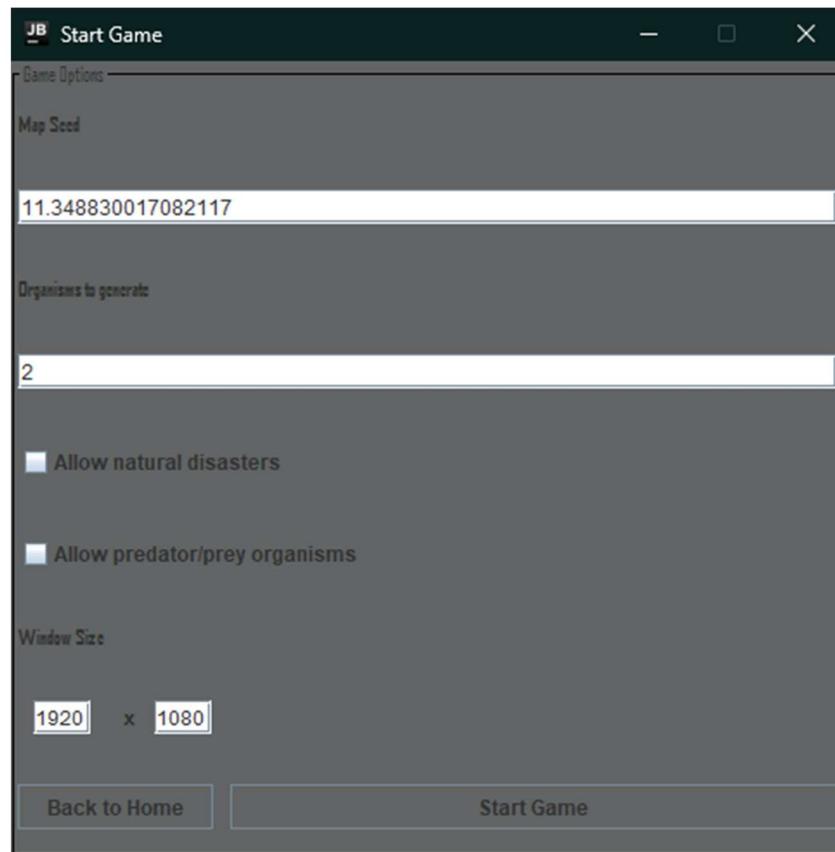


```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50517:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Number of organisms in organisms.json: 250
```

Once again, as expected, the program accepts this data and generates 250 organisms.

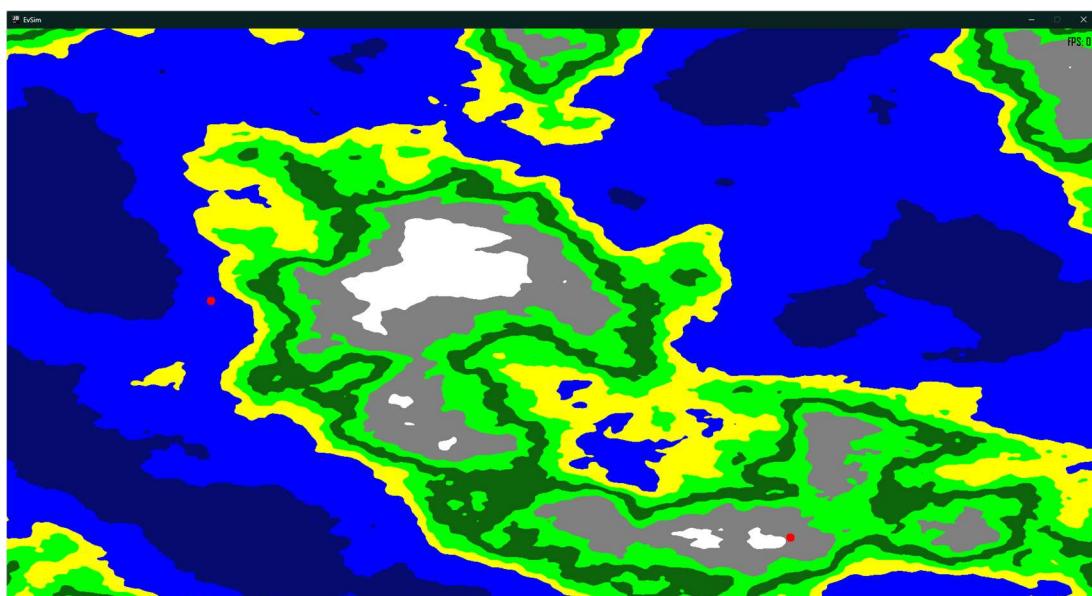


Boundary data will now be tested, valued of 2 and 300 should both be accepted. The decision to cap the amount of organisms that spawn to be set at 300, is that it can become too hectic with so many organisms on the screen, it may not be beneficial for the user to want to generate that many, and it will only slow down the processing in game.



```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50602:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Number of organisms in organisms.json: 2
```

As seen, 2 organisms have been accepted and generated.



JB Start Game

Game Options

Map Seed

11.348830017082117

Organisms to generate

300

Allow natural disasters

Allow predator/prey organisms

Window Size

1920 x 1080

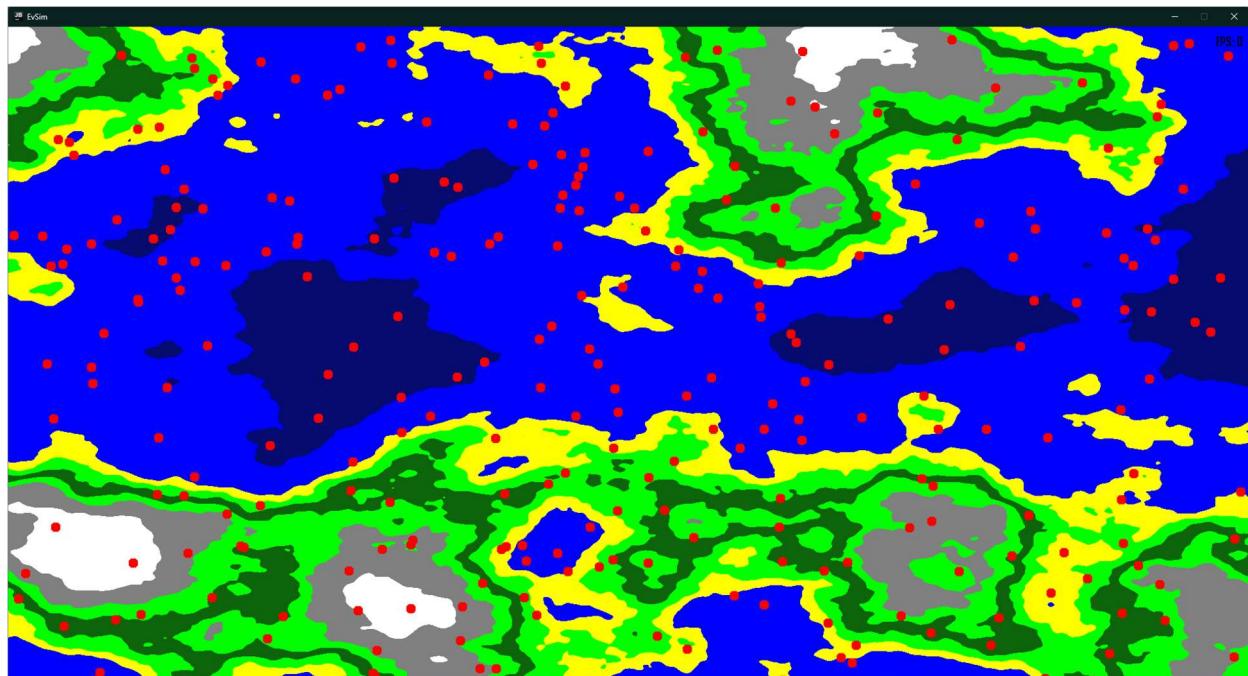
[Back to Home](#) [Start Game](#)

```

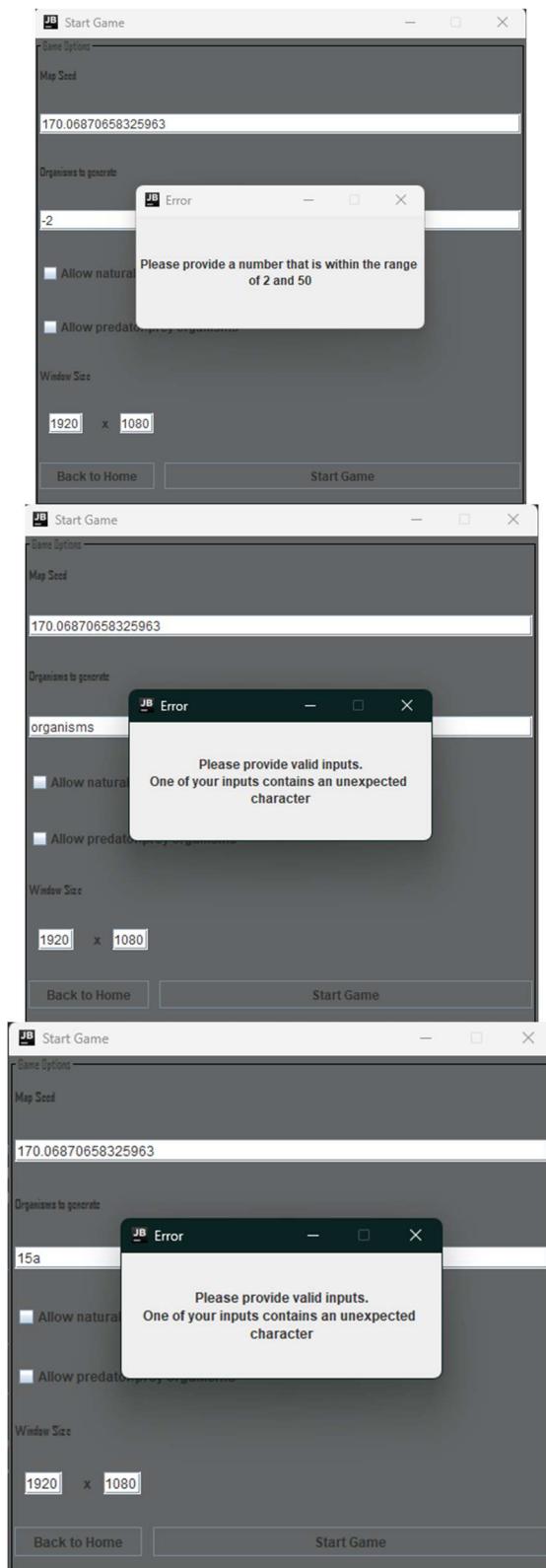
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50611:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Number of organisms in organisms.json: 300

```

Once again, this is accepted, and 300 organisms have been generated. As viewed below, 300 organisms becomes hectic for a user to manage, hence the cap at 300, as anything above would not be easy for the user to visualise evolution.



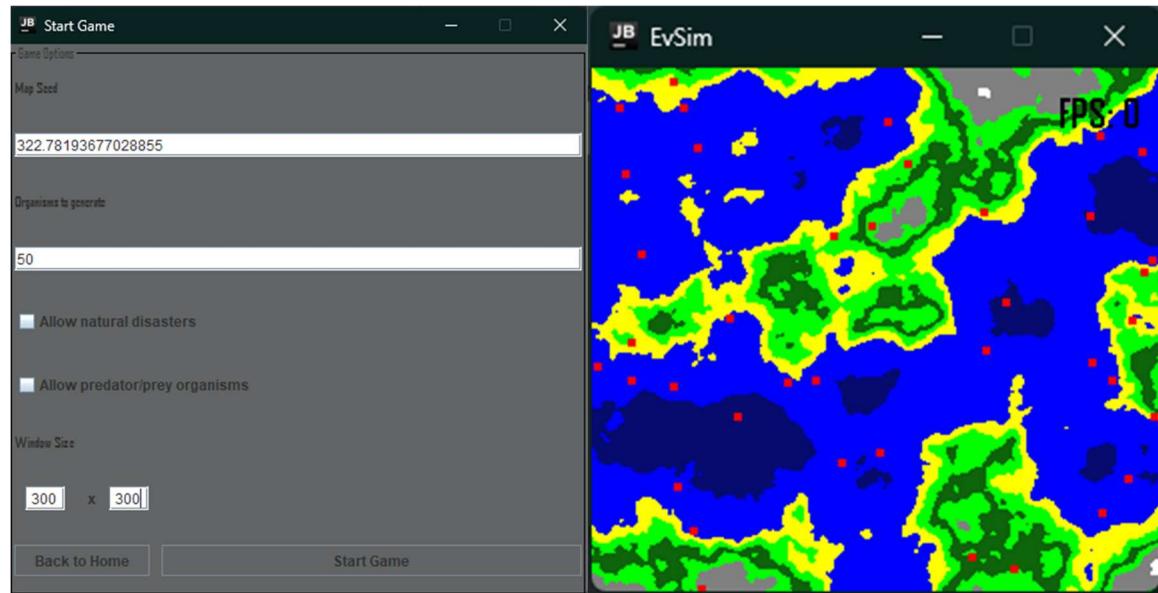
Now for the erroneous data, this should not be accepted by the program, and should provide the user with an appropriate error message.



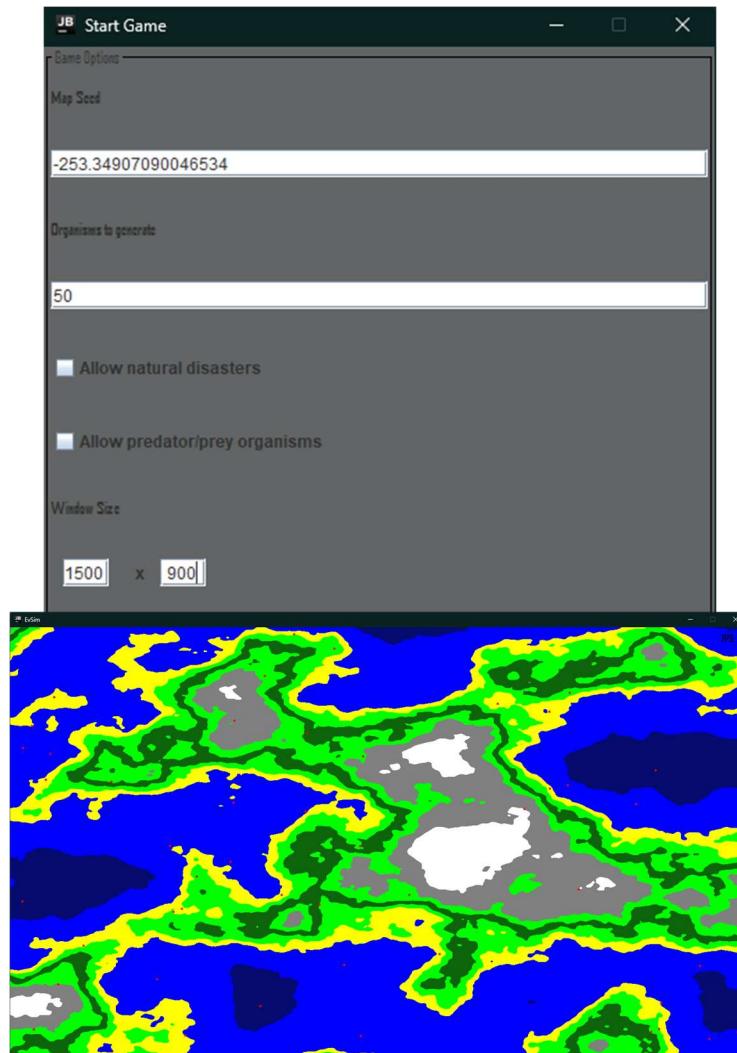
As expected, the program rejects the erroneous data, and provides the user with incentive to change their inputs.

3.1.4.3.3 Testing a1920TextField and a1080TextField

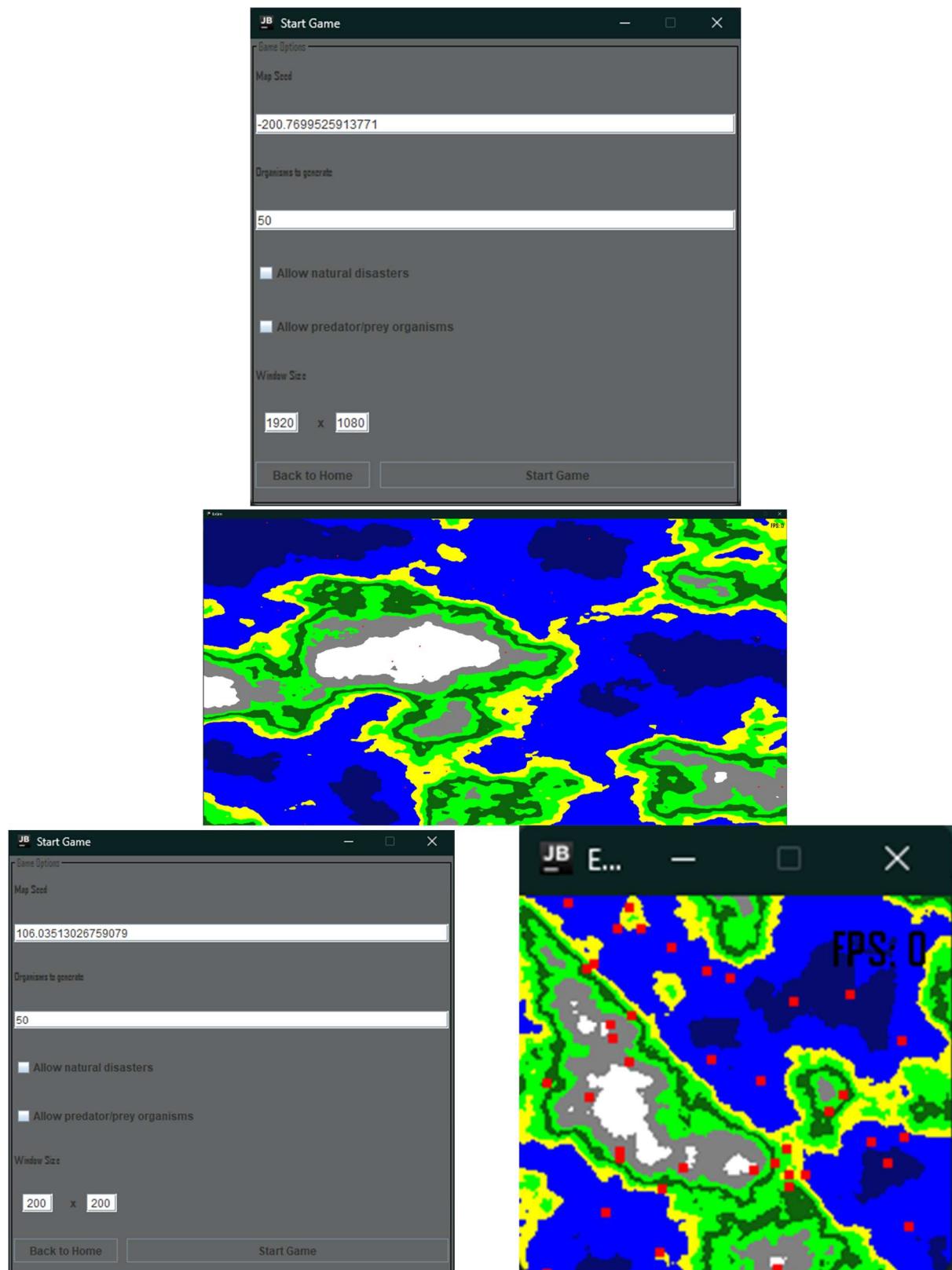
For the expected and boundary data, I can test both a1920TextField and a1080TextField. This is because they both relate to the window size and therefore can be tested in union.



As seen, the expected data works, displaying the screen in a 300 x 300 window. Testing 1500 x 900, we can see a similar result.



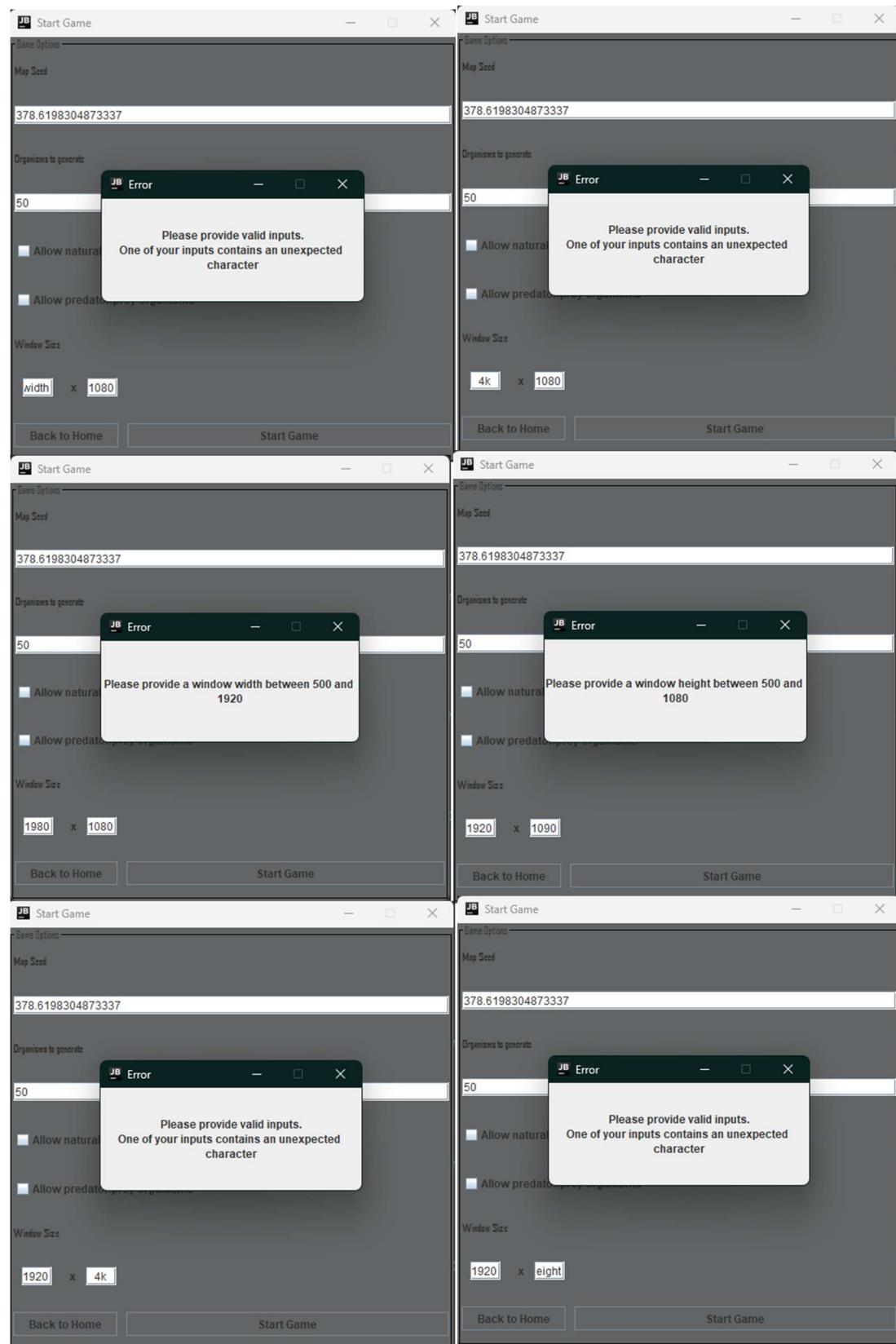
Boundary data can also be tested simultaneously, with the windows being generated no issues.



With this window being incredibly small, it would be impossible for EvSim to be used therefore I am going to **increase the bounds to 500 x 500**. This may change down the line, depending on the needs of the program. For example, if the side menu is not useable, the boundaries can be increased.

The program did accept the boundary data as valid data however, meaning it works.

For the erroneous data, the program should tell the user to change the width or height of the window, as seen below.



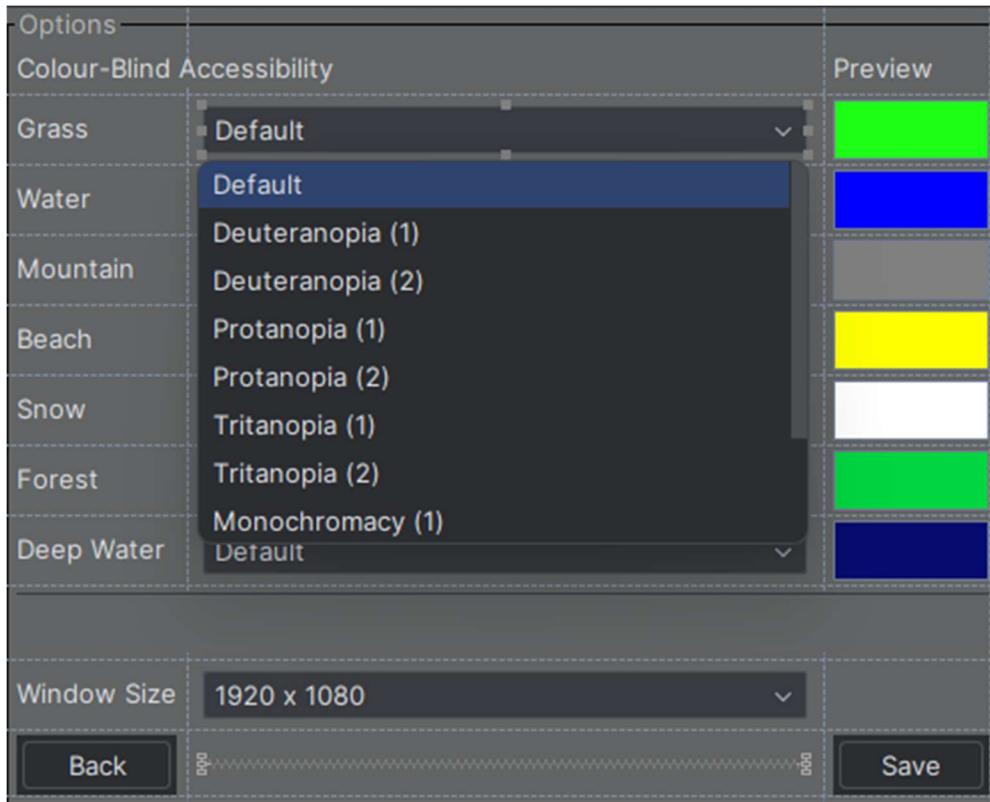
As expected, all the erroneous data is rejected by the program, prompting the user to change their inputs.

3.1.5 Main menu options

This menu will allow the user to change global settings, for example the colour scheme (see 2.4 Usability features – colour blind accessibility), and default window size.

3.1.5.1 Main menu options design

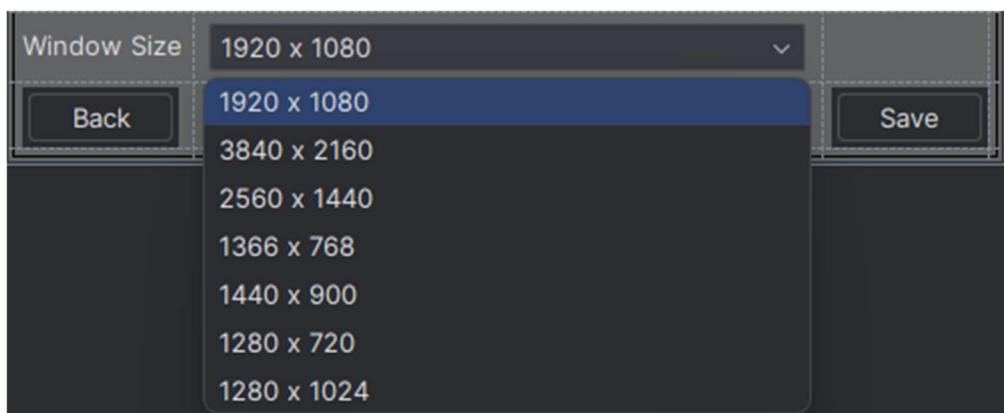
Due to the accessibility and general useability that the IntelliJ forms provided previously, I have chosen to use the same method for this window.



The plan for this menu is to offer the user the ability to change each biomes colour. This needs to be intuitive. The image below represents the design for this options menu, each biome is labelled on the left hand side, allowing the user to select from a list of colour blind friendly options, and a preview of that colour on the right hand side.

This menu also offers the user to select the default window size, from a list of common monitor sizes:

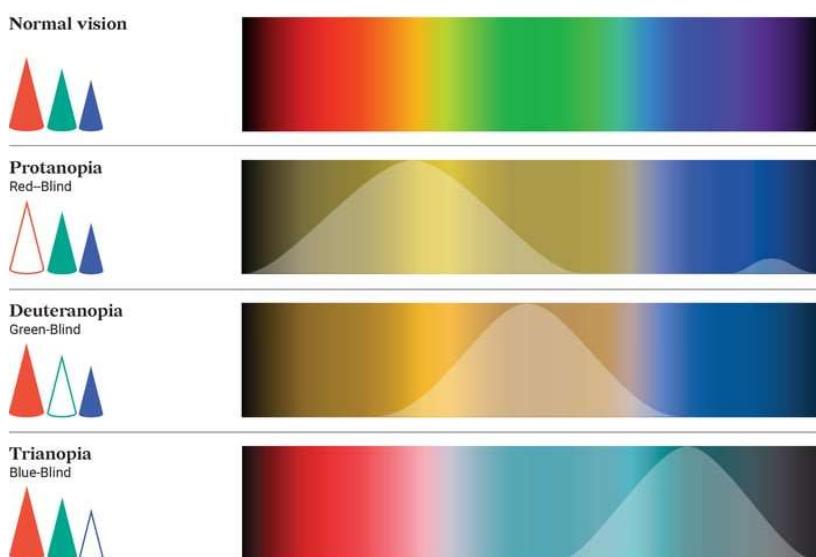
With the design complete, functionality now needs to be added. For the default window size to change, the user will have to restart the program, however the colour accessibility can be changed without a game restart, as the game is not yet started.



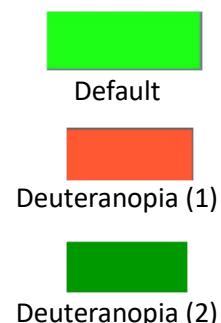
3.1.5.2 Adding functionality to the options menu

Similar to the start game menu, information can be retrieved from each of the input sections, as well as set. This means that the program can get the setting selected, and make the appropriate to the screen, for example the colour options, which require constant updates. This will require an item listener, as the component is an item.

For the colour-blind colours themselves, I am referring to this chart. Taking the normal colour, I will choose an alternative which has attributes that benefit someone with colour blindness. For example colour contrast helps significantly for those with colour blindness, therefore colours with a higher contrast will be selected.



For this example, I will take the default grass colour within EvSim, and show the alternative colours that are available. See below.



The user will have the choice of 11 colours per biome, which

may improve their ability to differentiate the biomes. The colours can be alternated per biome, meaning even if the presets aren't perfect, the user can change it as they please.

Making this feature functional, as mentioned will require the itemListener class, specifically itemStateChanged. Using this class, I will compare the item which was changed, take the state that it was changed to, and then make it display that colour in the preview section. Due to the preview section being a separate JPanel, to change the colour, I can use .setBackground(Color). The code for such as shown below, for this example it will only show for Deuteranopia.

```

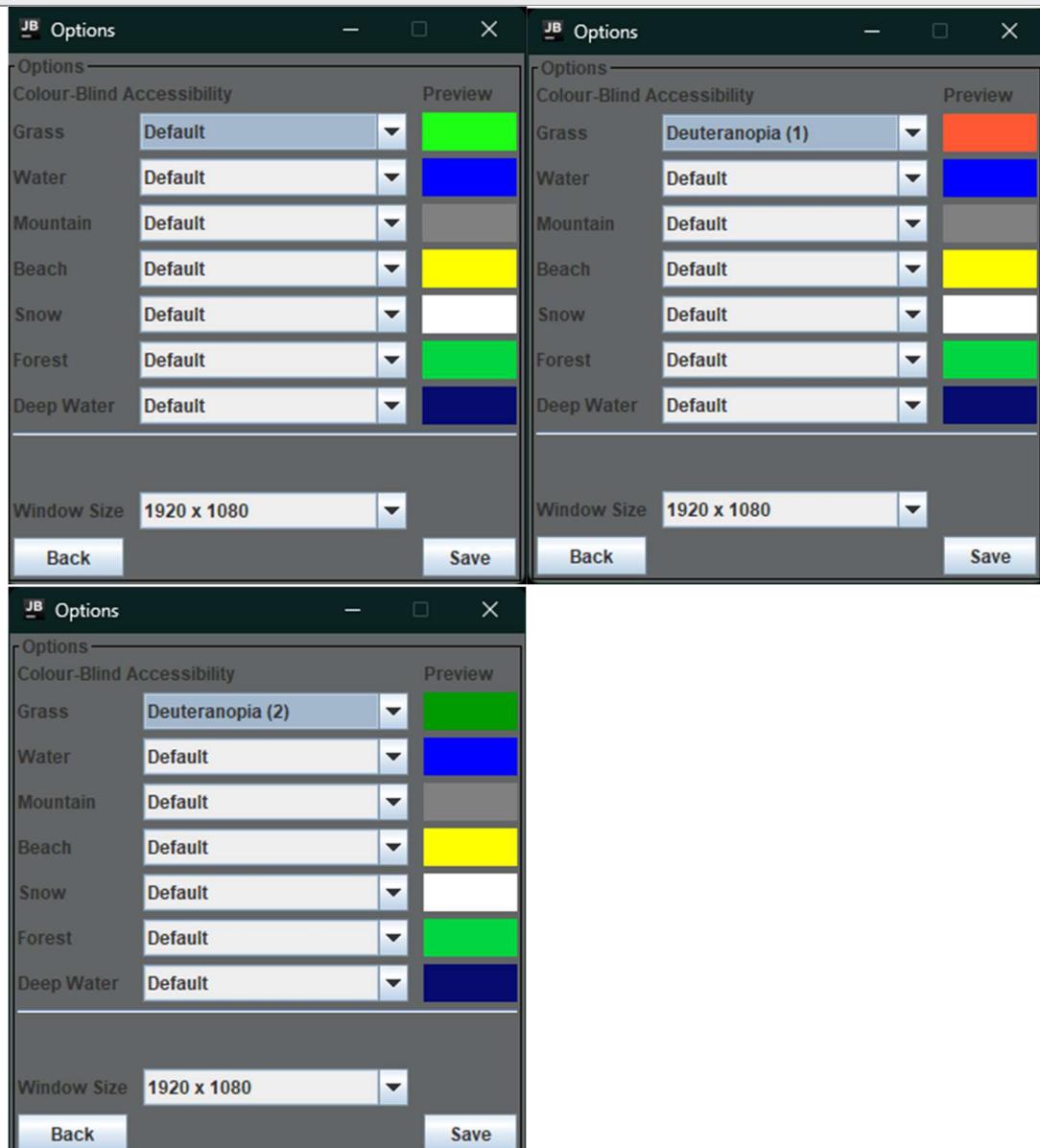
1.   public mainMenuOptions(){
2.       grassColour.addItemListener(this); // Adds the item listener
3.
4.       optionsFrame = new JFrame("Options");
5.       optionsFrame.setPreferredSize(new Dimension(357,388)); // Same width as in the form
6.
7.       optionsFrame.add(mainMenu); // As the panel already contains the labels and drop-downs,
8.       // only the panel has to be added to the frame.
9.       optionsFrame.setResizable(false);
10.      optionsFrame.pack();
11.      optionsFrame.setLocationRelativeTo(null);
12.      optionsFrame.setVisible(true);
13.
14.     @Override
15.     public void itemStateChanged(ItemEvent e) {
16.         if(e.getSource() == grassColour){
17.             if(grassColour.getSelectedItem() == "Default")
18.                 grassPreview.setBackground(Color.GREEN); // If unchanged, set the grass colour to green
19.             else if(grassColour.getSelectedItem() == "Deuteranopia (1)")
20.                 grassPreview.setBackground(new Color(255, 88, 51));
21.         }
22.     }
23. }
```

```

18.         else if (grassColour.getSelectedItem() == "Deuteranopia (2)")
grassPreview.setBackground(new Color(0, 153, 0));
19.     }
20. }

```

Java



As shown, this correctly updates the preview section. I would now need to update this within mapGenerator.java in order for it to function as intended. The updated code for this is below.

```

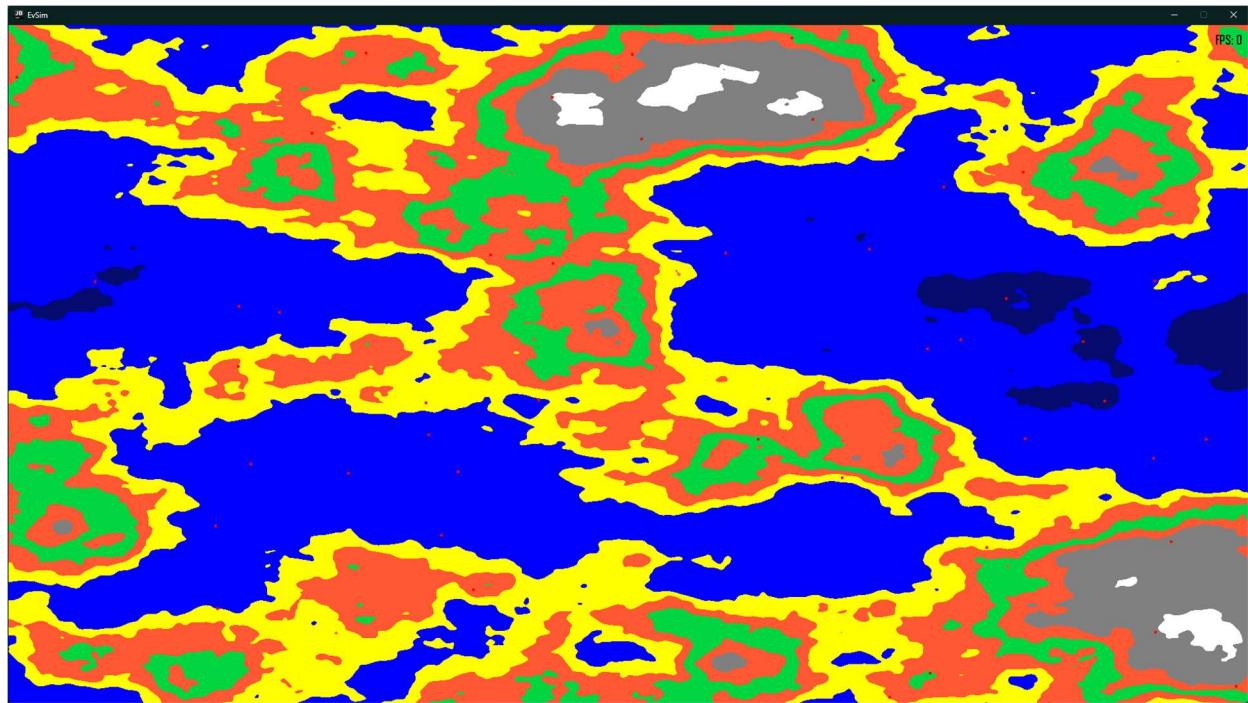
1.     public void itemStateChanged(ItemEvent e) {
2.         if(e.getSource() == grassColour){
3.             if(grassColour.getSelectedItem() == "Default")
grassPreview.setBackground(Color.GREEN);
4.             else if(grassColour.getSelectedItem() == "Deuteranopia (1)")
grassPreview.setBackground(new Color(255, 88, 51));
5.             else if (grassColour.getSelectedItem() == "Deuteranopia (2)")
grassPreview.setBackground(new Color(0, 153, 0));
6.             generateMap.terrainColors[0] = grassPreview.getBackground();
7.         }
8.     }

```

Java

The addition in this code which causes the map to update, is line 6, where the terrain colour is set to whatever colour is held in the preview panel. If the preview panel is not updated, then the colour that is

generated stays default, as the panel is set to that by default. Testing this in game for Deuteranopia (1) produces the following results.



The grass texture is now set to a colour that people who have Deuteranopia can view easily. One issue that can be seen here is that the organisms are now not easily divisible from the grass. This will be fixed later in development, as the appearance of the organisms is finalised. For now, the organism only needs to appear as a red dot, as the processing behind is more important to focus on.

To finish this section of the options menu, the other options will now be added. See below for the final code.

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.ItemEvent;
4. import java.awt.event.ItemListener;
5. import java.awt.event.MouseEvent;
6. import java.awt.event.MouseListener;
7.
8. public class mainMenuOptions implements ItemListener, MouseListener {
9.     private JPanel mainMenu;
10.    private JComboBox grassColour;
11.    private JComboBox waterColour;
12.    private JComboBox mountainColour;
13.    private JComboBox beachColour;
14.    private JComboBox snowColour;
15.    private JComboBox forestColour;
16.    private JComboBox deepWaterColour;
17.    private JComboBox windowSize;
18.    private JButton backButton;
19.    private JButton saveButton;
20.    private JPanel grassPreview;
21.    private JPanel waterPreview;
22.    private JPanel mountainPreview;
23.    private JPanel beachPreview;
24.    private JPanel snowPreview;
25.    private JPanel forestPreview;
26.    private JPanel deepWaterPreview;
27.    JFrame optionsFrame;
28.    public mainMenuOptions(){
29.        grassColour.addItemListener(this); // Adds the item listener
```

```

30.         waterColour.addItemListener(this);
31.         mountainColour.addItemListener(this);
32.         beachColour.addItemListener(this);
33.         forestColour.addItemListener(this);
34.         deepWaterColour.addItemListener(this);
35.
36.         optionsFrame = new JFrame("Options");
37.         optionsFrame.setPreferredSize(new Dimension(357,388)); // Same width as in the form
creator
38.         optionsFrame.add(mainMenu); // As the panel already contains the labels and drop-
downs, only the panel has to be added to the frame.
39.         optionsFrame.setResizable(false);
40.         optionsFrame.pack();
41.         optionsFrame.setLocationRelativeTo(null);
42.         optionsFrame.setVisible(true);
43.     }
44.
45.     @Override
46.     public void itemStateChanged(ItemEvent e) {
47.         if(e.getSource() == grassColour){ // If the user changed the grass drop down
48.             if(grassColour.getSelectedItem() == "Default")
grassPreview.setBackground(Color.GREEN); // Sets the colour for each colour-blindness
49.                 else if(grassColour.getSelectedItem() == "Deutanopia (1)")
grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
50.                 else if (grassColour.getSelectedItem() == "Deutanopia (2)")
grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
51.                 else if (grassColour.getSelectedItem() == "Protanopia (1)")
grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
52.                 else if (grassColour.getSelectedItem() == "Protanopia (2)")
grassPreview.setBackground(new Color(51, 51, 204)); // Purple-blue
53.                 else if (grassColour.getSelectedItem() == "Tritanopia (1)")
grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
54.                 else if (grassColour.getSelectedItem() == "Tritanopia (2)")
grassPreview.setBackground(new Color(255, 217, 0)); // Yellow
55.                 else if (grassColour.getSelectedItem() == "Monochromacy (1)")
grassPreview.setBackground(new Color(0, 0, 0)); // Black
56.                 else if (grassColour.getSelectedItem() == "Monochromacy (2)")
grassPreview.setBackground(new Color(255, 255, 255)); // White
57.                 else if (grassColour.getSelectedItem() == "Trichromacy (1)")
grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
58.                 else if (grassColour.getSelectedItem() == "Trichromacy (2)")
grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
59.                 generateMap.terrainColors[0] = grassPreview.getBackground(); // Sets the grass
(first index) to be the colour of the panel.
60.             } else if (e.getSource() == waterColour){
61.                 if(waterColour.getSelectedItem() == "Default")
waterPreview.setBackground(Color.BLUE); // Sets the colour for each colour-blindness
62.                     else if(waterColour.getSelectedItem() == "Deutanopia (1)")
waterPreview.setBackground(new Color(0, 128, 128)); // Teal
63.                     else if(waterColour.getSelectedItem() == "Deutanopia (2)")
waterPreview.setBackground(new Color(255, 0, 0)); // Red
64.                     else if(waterColour.getSelectedItem() == "Protanopia (1)")
waterPreview.setBackground(new Color(0, 128, 128)); // Teal
65.                     else if(waterColour.getSelectedItem() == "Protanopia (2)")
waterPreview.setBackground(new Color(255, 0, 0)); // Red
66.                     else if(waterColour.getSelectedItem() == "Tritanopia (1)")
waterPreview.setBackground(new Color(0, 128, 128)); // Teal
67.                     else if(waterColour.getSelectedItem() == "Tritanopia (2)")
waterPreview.setBackground(new Color(255, 255, 0)); // Yellow
68.                     else if(waterColour.getSelectedItem() == "Monochromacy (1)")
waterPreview.setBackground(new Color(0, 0, 50)); // Dark Blue/Black
69.                     else if(waterColour.getSelectedItem() == "Monochromacy (2)")
waterPreview.setBackground(new Color(150, 150, 150)); // Light Grey
70.                     else if(waterColour.getSelectedItem() == "Trichromacy (1)")
waterPreview.setBackground(new Color(0, 128, 128)); // Teal
71.                     else if(waterColour.getSelectedItem() == "Trichromacy (2)")
waterPreview.setBackground(new Color(255, 0, 0)); // Red
72.                     generateMap.terrainColors[1] = waterPreview.getBackground(); // Sets the water
(second index) to be the colour of the panel
73.             } else if (e.getSource() == mountainColour){

```

```

74.             if(mountainColour.getSelectedItem() == "Default")
mountainPreview.setBackground(Color.GRAY); // Sets the colour for each colour-blindness
75.             else if(mountainColour.getSelectedItem() == "Deuteranopia (1)")
mountainPreview.setBackground(new Color(96, 96, 96)); // All of these are shades of grey (to avoid
biome blending), the higher the number in each, the lighter the grey
76.             else if(mountainColour.getSelectedItem() == "Deuteranopia (2)")
mountainPreview.setBackground(new Color(160, 160, 160));
77.             else if(mountainColour.getSelectedItem() == "Protanopia (1)")
mountainPreview.setBackground(new Color(50, 50, 50));
78.             else if(mountainColour.getSelectedItem() == "Protanopia (2)")
mountainPreview.setBackground(new Color(150, 150, 150));
79.             else if(mountainColour.getSelectedItem() == "Tritanopia (1)")
mountainPreview.setBackground(new Color(60, 60, 60));
80.             else if(mountainColour.getSelectedItem() == "Tritanopia (2)")
mountainPreview.setBackground(new Color(140, 140, 140));
81.             else if(mountainColour.getSelectedItem() == "Monochromacy (1)")
mountainPreview.setBackground(new Color(70, 70, 70));
82.             else if(mountainColour.getSelectedItem() == "Monochromacy (2)")
mountainPreview.setBackground(new Color(130, 130, 130));
83.             else if(mountainColour.getSelectedItem() == "Trichromacy (1)")
mountainPreview.setBackground(new Color(80, 80, 80));
84.             else if(mountainColour.getSelectedItem() == "Trichromacy (2)")
mountainPreview.setBackground(new Color(120, 120, 120));
85.             generateMap.terrainColors[2] = mountainPreview.getBackground(); // Sets the
mountain (third index) to be the colour of the panel
86.         } else if(e.getSource() == beachColour){
87.             if(beachColour.getSelectedItem() == "Default")
beachPreview.setBackground(Color.YELLOW); // Sets the colour for each colour-blindness
88.             else if(beachColour.getSelectedItem() == "Deuteranopia (1)")
beachPreview.setBackground(new Color(128, 128, 0)); // yellow-brown
89.             else if(beachColour.getSelectedItem() == "Deuteranopia (2)")
beachPreview.setBackground(new Color(0, 0, 255)); // light blue
90.             else if(beachColour.getSelectedItem() == "Protanopia (1)")
beachPreview.setBackground(new Color(128, 100, 0)); // orange-brown
91.             else if(beachColour.getSelectedItem() == "Protanopia (2)")
beachPreview.setBackground(new Color(0, 0, 200)); // dark blue
92.             else if(beachColour.getSelectedItem() == "Tritanopia (1)")
beachPreview.setBackground(new Color(100, 100, 0)); // olive green
93.             else if(beachColour.getSelectedItem() == "Tritanopia (2)")
beachPreview.setBackground(new Color(50, 50, 200)); // Blue-purple
94.             else if(beachColour.getSelectedItem() == "Monochromacy (1)")
beachPreview.setBackground(new Color(90, 90, 90)); // dark grey
95.             else if(beachColour.getSelectedItem() == "Monochromacy (2)")
beachPreview.setBackground(new Color(170, 170, 170)); // Light grey
96.             else if(beachColour.getSelectedItem() == "Trichromacy (1)")
beachPreview.setBackground(new Color(100, 100, 50)); // Brown
97.             else if(beachColour.getSelectedItem() == "Trichromacy (2)")
beachPreview.setBackground(new Color(100, 50, 255)); // Purple
98.             generateMap.terrainColors[3] = beachPreview.getBackground(); // Sets the beach
(fourth index) to be the colour of the panel
99.         } else if(e.getSource() == forestColour) {
100.             if(forestColour.getSelectedItem() == "Default") forestPreview.setBackground(new
Color(13, 100, 10)); // Sets the colour for each colour-blindness
101.             else if(forestColour.getSelectedItem() == "Deuteranopia (1)")
forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
102.             else if(forestColour.getSelectedItem() == "Deuteranopia (2)")
forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
103.             else if(forestColour.getSelectedItem() == "Protanopia (1)")
forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
104.             else if(forestColour.getSelectedItem() == "Protanopia (2)")
forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
105.             else if(forestColour.getSelectedItem() == "Tritanopia (1)")
forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
106.             else if(forestColour.getSelectedItem() == "Tritanopia (2)")
forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
107.             else if(forestColour.getSelectedItem() == "Monochromacy (1)")
forestPreview.setBackground(new Color(15, 15, 15)); // Lighter shade of black
108.             else if(forestColour.getSelectedItem() == "Monochromacy (2)")
forestPreview.setBackground(new Color(205, 180, 255)); // Lilac
109.             else if(forestColour.getSelectedItem() == "Trichromacy (1)")
forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey

```

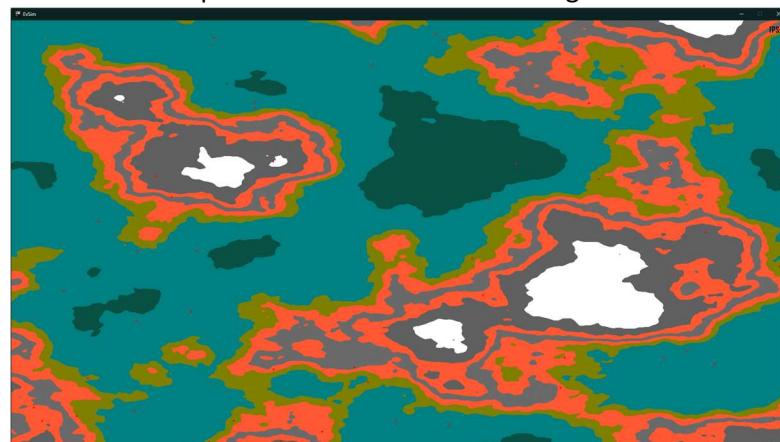
```

110.             else if(forestColour.getSelectedItem() == "Trichromacy (2)")
forestPreview.setBackground(new Color(150, 150, 150)); // Lighter Grey
111.             generateMap.terrainColors[5] = forestPreview.getBackground(); // Sets the forest
(fifth index) to be the colour of the panel
112.         } else if(e.getSource() == deepWaterColour) {
113.             if(deepWaterColour.getSelectedItem() == "Default")
deepWaterPreview.setBackground(new Color(6, 11, 112)); // Sets the colour for each colour-blindness
114.             else if(deepWaterColour.getSelectedItem() == "Deutanopia (1)")
deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
115.             else if(deepWaterColour.getSelectedItem() == "Deutanopia (2)")
deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
116.             else if(deepWaterColour.getSelectedItem() == "Protanopia (1)")
deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
117.             else if(deepWaterColour.getSelectedItem() == "Protanopia (2)")
deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
118.             else if(deepWaterColour.getSelectedItem() == "Tritanopia (1)")
deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
119.             else if(deepWaterColour.getSelectedItem() == "Tritanopia (2)")
deepWaterPreview.setBackground(new Color(255, 255, 0)); // Yellow
120.             else if(deepWaterColour.getSelectedItem() == "Monochromacy (1)")
deepWaterPreview.setBackground(new Color(30, 30, 30)); // Very dark grey
121.             else if(deepWaterColour.getSelectedItem() == "Monochromacy (2)")
deepWaterPreview.setBackground(new Color(180, 255, 200)); // Mint Green
122.             else if(deepWaterColour.getSelectedItem() == "Trichromacy (1)")
deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
123.             else if(deepWaterColour.getSelectedItem() == "Trichromacy (2)")
deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
124.             generateMap.terrainColors[6] = deepWaterPreview.getBackground(); // Sets the deep
water (sixth index to be the colour of the panel
125.         }
126.         // SNOW is white so does not need to be changed
127.     }

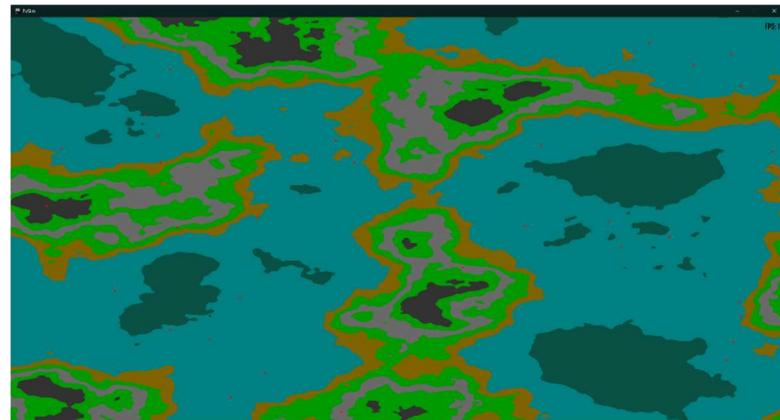
```

Java

Refer to comments for explanation of each section. Comments are marked in red, starting with a //
Some test examples of this can be seen working below.



Deutanopia (1)



Protanopia (1)

The benefit of allowing the user to change each biome individually means that they can alternate the colours on the side to whatever works best for them, in the event that the selections are not clear enough. This is the benefit of allowing a preview section next to the drop-down.

This next section covers the functionality of the window size. This will need to check the monitor size of the user, so that they cannot set the window size to be greater than what they can see, for example a user with a 1920 x 1080 screen should not be able to set their window size to 3840 x 2160. This window size change will have to update Main.java, as the values set here are passed all throughout the program.

In order to make changes permanent, the changes will have to be physically stored, therefore I am going to make another JSON file, called options.json, which can therefore be read and used across the program. The design for the default settings are displayed within the code block below.

```

1. {
2.     "WindowHeight": 1080,
3.     "WindowWidth": 1920,
4.     "Colours": {
5.         "0": "Color.GREEN", "1": "Color.BLUE", "2": "Color.GRAY", "3": "Color.YELLOW", "4": "Color.WHITE", "5": "new Color(13, 100, 10)", "6": "new Color(6, 11, 112)"
6.     }
7. }
```

JSON

This will therefore have to be read in both Main.java, and generateMap.java. I will be again using the same method for reading JSON as within 3.1.3.2. The code block below represents what the Main file executes upon start of the game.

```

1. import com.fasterxml.jackson.databind.JsonNode;
2. import com.fasterxml.jackson.databind.ObjectMapper;
3. import java.io.File;
4. public class Main {
5.     public static int windowHeight = 1920; // Default value
6.     public static int windowHeight = 1080; // Default value
7.     public static void main(String[] args) {
8.         ObjectMapper objectMapper = new ObjectMapper(); // Initialising the JSON Object Mapper
9.         try {
10.             JsonNode data = objectMapper.readTree(new File("options.json")); // Reads the JSON file as a Node (does not require a class)
11.             windowHeight = data.get("WindowWidth").asInt(); // Gets the value stored at the key, and returns as Integer (default is JsonNode)
12.             windowHeight = data.get("WindowHeight").asInt();
13.         } catch (Exception e) {
14.             e.printStackTrace();
15.         }
16.         OpeningMenu window = new OpeningMenu(windowHeight, windowHeight);
17.         window.startMenuThread();
18.     }
19. }
```

Java

As the comments explains, the program now opens the main window as whatever is set within the JSON file. If there is an error, or there is no present value within the JSON file, then the window opens as 1920 x 1080 (default). This now needs to be available to change through the options menu, which the section below explains.

Getting the user input for Window Size:

```

1. if(e.getSource() == windowSize){ // Takes the setting currently chosen in the menu, and updating the relating values
2.     if(windowSize.getSelectedItem() == "1920 x 1080"){
3.         newWindowHeight = 1080;
4.         newWindowWidth = 1920;
5.     } else if (windowSize.getSelectedItem() == "3840 x 2160") {
6.         newWindowHeight = 2160;
7.         newWindowWidth = 3840;
```

```

8.         } else if (windowSize.getSelectedItem() == "2560 x 1440") {
9.             newWindowHeight = 1440;
10.            newWindowWidth = 2560;
11.        } else if (windowSize.getSelectedItem() == "1366 x 768") {
12.            newWindowHeight = 768;
13.            newWindowWidth = 1366;
14.        } else if (windowSize.getSelectedItem() == "1440 x 900") {
15.            newWindowHeight = 900;
16.            newWindowWidth = 1440;
17.        } else if (windowSize.getSelectedItem() == "1280 x 720") {
18.            newWindowHeight = 720;
19.            newWindowWidth = 1280;
20.        } else if (windowSize.getSelectedItem() == "1280 x 1024") {
21.            newWindowHeight = 1024;
22.            newWindowWidth = 1280;
23.        }

```

Java

At the current stage, this does not change the window height, and does not also validate that the user's window is large enough to fit each setting, as this should be done when the user saves the settings. Adding in functionality to the save button, the screen data can now be validated. The functionality to the buttons is the same as within the start game options, if the mouse is clicked on the button, then validate the inputs (providing an error message if wrong), and perform the relevant actions.

Like in the main class, I will be using the object mapper class to update the JSON file each time the screen size is changed. This will allow the program to change size in accordance once it is restarted. The code for the buttons, as well as writing the new values to JSON (for window size) is below.

```

1.     @Override
2.     public void mouseClicked(MouseEvent e) {
3.         if(e.getSource() == saveButton){
4.             ObjectMapper objectMapper = new ObjectMapper(); //Initialising the object mapper
5.             if(newWindowHeight <= maxWindowHeight && newWindowWidth <= maxWindowWidth){ //Checking that the window size can fit
6.                 try {
7.                     JsonNode data = objectMapper.readTree(new File("options.json"));
8.                     ObjectNode newData = (ObjectNode) data; // Passing into a node (so it can be changed)
9.                     newData.put("WindowHeight", newWindowHeight); // Changing the values, first part is the key to change, second is what to change the value to
10.                    newData.put("WindowWidth", newWindowWidth);
11.                    String newJSON = objectMapper.writeValueAsString(newData); // Changing to a string, so it can be written to the file
12.                    FileWriter fileWriter = new FileWriter("options.json");
13.                    fileWriter.write(newJSON); // Writing to the file
14.                    fileWriter.close();
15.                    System.out.println("Max height: " + maxWindowHeight + " Max Width: " + maxWindowWidth + " New Height: " + newWindowHeight + " New Width: " + newWindowWidth); // Checking data
16.                } catch (IOException ex) {
17.                    throw new RuntimeException(ex);
18.                }
19.            } else {
20.                ErrorWindow newError = new ErrorWindow("<html><p style=\"text-align:center;\">That size is not available for your monitor.<br>Please select a smaller size</p></html>");
21.                // if the window size is larger than the screen, then the appropriate error message is displayed
22.            }
23.        }
24.        if(e.getSource() == backButton) {
25.            optionsFrame.dispose(); // Ignores any inputs, and closes the menu
26.        }
27.    }

```

Java

All that is left to add now is the changing of colours, this will be added into the section above, under line 10. See below for the code to be added.

The original plan to have JSON represent the raw Java line, immediately proved to be a problem. As the colours change so frequently, specifically their RGB values, representing the colours as `Color.BLUE` for example would not written or read easily. That value can only be stored as a string in JSON, meaning the literal Java attribute cannot be read within the map generation. Therefore the JSON file would have to store the Red, Green, and Blue for each colour. The best way to represent this is an object array, or an `ArrayNode` in Jackson. This could then be added under the key of “Colours” within `options.json`.

```

1.     ArrayNode newColours = objectMapper.createArrayNode();
2.     for (Color color : terrainColours) { // For each colour in terrainColours
3.         ObjectNode colorNode = objectMapper.createObjectNode(); // A new Object
4.         colorNode.put("red", color.getRed()); // Adding the red, green and blue values for each
colours into individual objects
5.         colorNode.put("green", color.getGreen());
6.         colorNode.put("blue", color.getBlue());
7.         newColours.add(colorNode); // Adding it to the object array
8.     }
9.
10.    newData.put("Colours", newColours); // Writing to the new JSON

```

Java

This saves the JSON in the following format, which means each colour can now be read as red, green, and blue values.

```

1. {
2.     "WindowHeight": 1080,
3.     "WindowWidth": 1920,
4.     "Colours": [
5.         {
6.             "red": 0,
7.             "green": 255,
8.             "blue": 0
9.         },
10.        {
11.            "red": 0,
12.            "green": 0,
13.            "blue": 255
14.        },
15.        {
16.            "red": 128,
17.            "green": 128,
18.            "blue": 128
19.        },
20.        {
21.            "red": 255,
22.            "green": 255,
23.            "blue": 0
24.        },
25.        {
26.            "red": 255,
27.            "green": 255,
28.            "blue": 255
29.        },
30.        {
31.            "red": 13,
32.            "green": 100,
33.            "blue": 10
34.        },
35.        {
36.            "red": 6,
37.            "green": 11,
38.            "blue": 112
39.        }
40.    ]
41. }

```

JSON

Within the map generation class, I have also added `readJSONColours()`, which takes the colours from the JSON file, and updates the local `terrainColours` array.

```

1.     private void readJSONColours(){
2.         ObjectMapper objectMapper = new ObjectMapper() // Initialising the JSON Object Mapper
3.         try {
4.             JsonNode data = objectMapper.readTree(new File("options.json")); // Reads the JSON
file as a Node (does not require a class)
5.             JsonNode colour = data.get("Colours");
6.             int counter = 0;
7.             for (JsonNode colours : colour) {
8.                 terrainColors[counter] = new Color(colours.get("red").asInt(),
colours.get("green").asInt(), colours.get("blue").asInt());
9.                 counter++;
10.            }
11.        } catch (Exception e) {
12.            e.printStackTrace();
13.        }
14.    }

```

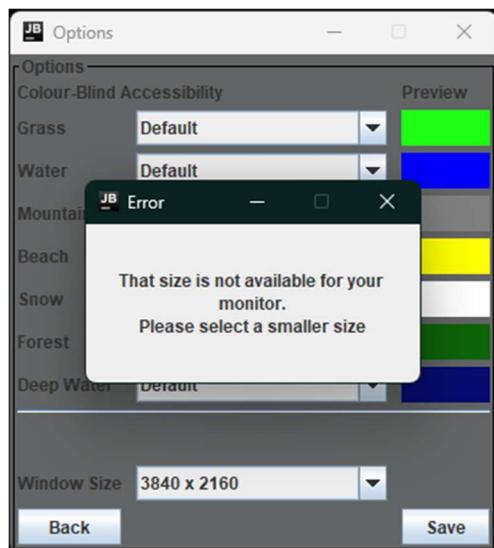
Java

All that is left now is to test this, to ensure that the options menu is fully functional.

3.1.5.3 Testing the main menu options panel

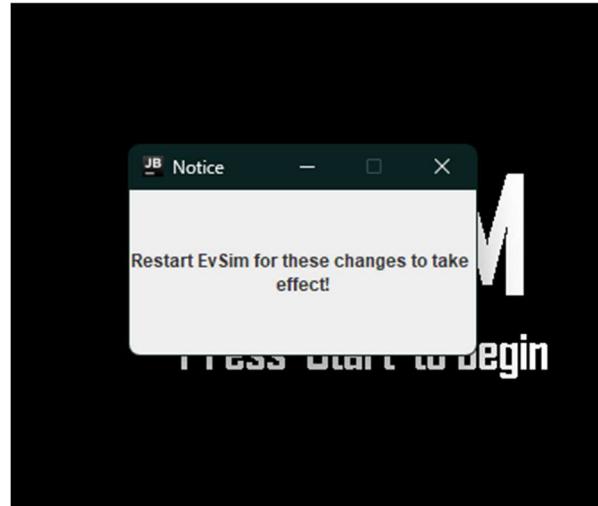
For this test to be successful, the options which are selected within the panel should make the change to the program either upon restart (window size), or when the simulation is started (colours).

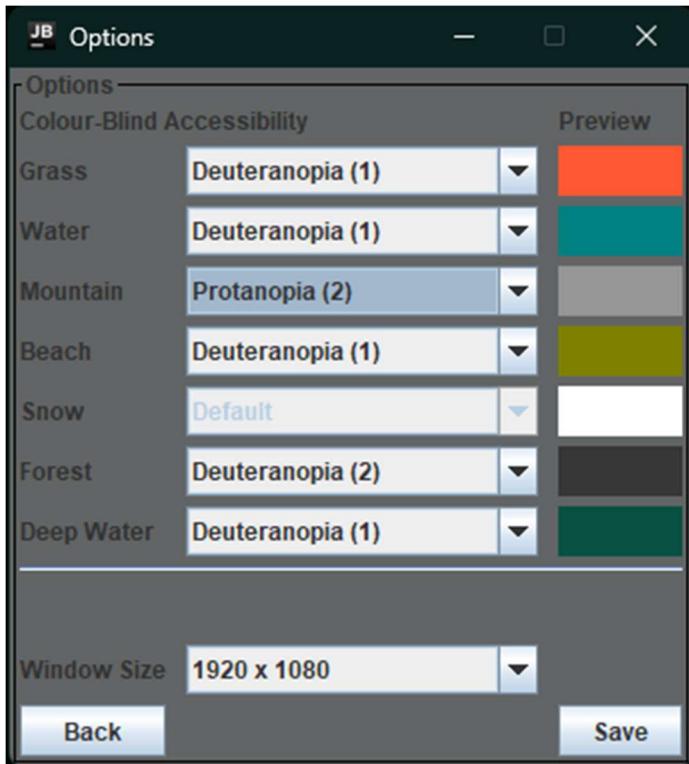
For this one, I will provide screenshots of the effect that the options panel has.



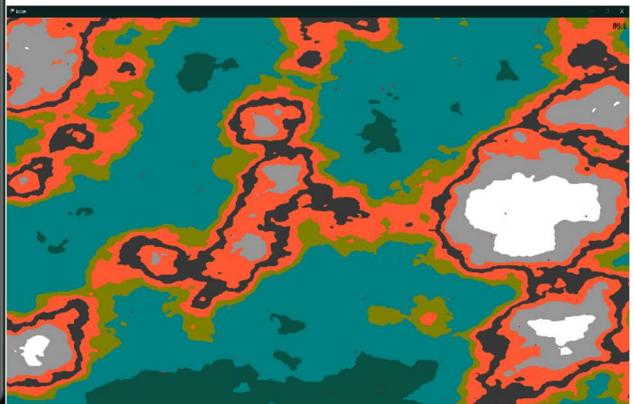
This first example shows what would happen if the user attempts to set the window size as something larger than their monitor. Upon the user attempting to save, an error window is displayed, letting the user know that they cannot select this option, and how they should select a smaller window size. The erroneous option cannot be saved physically, avoiding the window taking up too much size on the user's screen.

This example shows what is displayed to the user if they decide to change to an accepted monitor size. The options menu closes, and a notice window appears, instructing the user to restart the program to make any changes take effect. Despite this only showing when the window size is changed, if the colours are changed as well, they will be carried on even if the program is restarted.





This next section tests the colour change, upon the press of save, the program will accept these changes, saving them physically so whenever the user chooses to start the simulation, they will see these alterations. When starting the game, it is clear that these changes have taken place successfully.



This marks the end of the development of the front-end UI. There are few features left to develop for front-end UI, however it is reliant on the back-end (JSON) data being finished. For example, the side menu within the simulation needing to pull information from the constantly updating JSON files. The layout of these files are not finalised until back-end processing is complete, therefore it would not be efficient to build the menu beforehand. The next section within development will now cover the processing during the simulation, and how this is saved to the back end.

3.2 Calculations and processing

3.2.1 Time

Following closely with the 2.1 Problem decomposition, we start calculations and processing with the implementation of time. Time is key to EvSim's processing, as it covers the scale of evolution, how often the organisms move and update. Evidently this section is crucial for the latter success of Evolution.

To test time within EvSim, I will implement the theory of years. Each year for example will be 10 seconds in real life, with the game running at 60fps, there should be 600 frames per year. Every 600 frames, the year will increase, as well as the evolution of organisms. The age of the organisms will count the frames it is alive, and for every 600 frames, it will grow a year older. This means that the amount of 'years' the game has ran, and the amount of time a specific organism have been alive are separate variables. Using years within the program also allows for simulation speed to be increased or decreased by the user. With a fixed rate of 600 frames per year (60 fps, 10 seconds), if the user desires to increase the speed of the simulation, they will increase the fps, and therefore the amount of time taken for 600 frames to be reached. If the user increases the simulation speed by 2x, the game will run at 120fps, and each year will be 5 seconds. On the other hand, if a user decreases the simulation speed by half, then the game will run at 30fps and take 20 seconds per year. Organism evolution can therefore take place dependant on the frames that take place, for example the organism will move every 50 frames, but evolve every 1200 frames (2 years). These examples are unlikely to be the final values, as evolution cannot be finalised until implemented and tested.

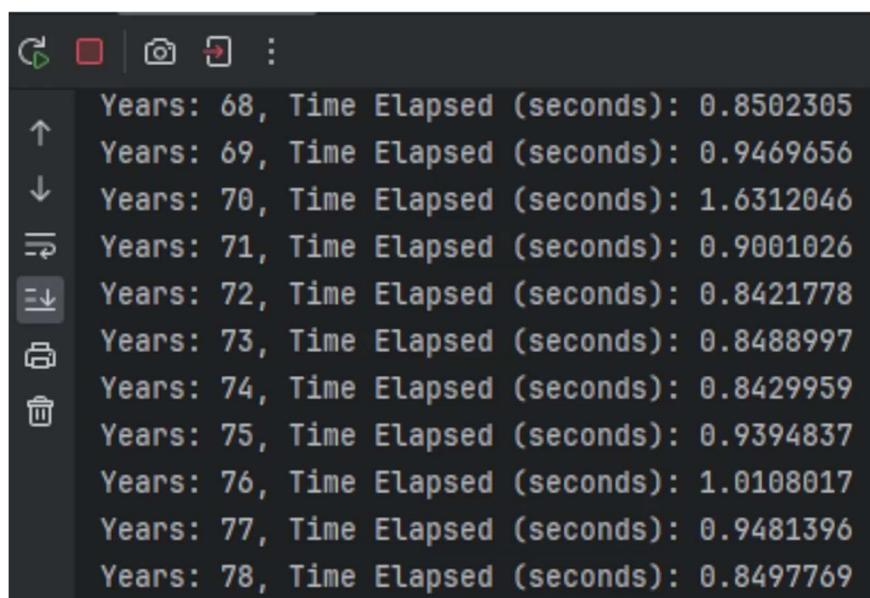
Below is a method used commonly within Java projects to calculate and use FPS within a program.

```
1. public void run() {
2.     double drawInterval = (double) 1000000000 / FPS; // Time interval between desired
frames (currently set to 60)
3.     double delta = 0;
4.     long lastTime = System.nanoTime();
5.     long currentTime;
6.     long timer = 0;
7.     long drawCount = 0;
8.
9.
10.    while (thread != null) {
11.        currentTime = System.nanoTime();
12.        delta += (currentTime - lastTime) / drawInterval; // Time since last frame and the
updated delta
13.        timer += (currentTime - lastTime); // time since last frame
14.        lastTime = currentTime; // update last time
15.        if(delta >= 1){ // checking when to repaint and update
16.            update();
17.            repaint();
18.            delta--;
19.            drawCount++; // increase to count fps
20.        }
21.        if(timer >= 1000000000) { // checking to see if one second has passed
22.            currentFPS = drawCount; //The current fps to use
23.            drawCount = 0;
24.            timer = 0; // resetting variables
25.        }
26.    }
27. }
```

Java

Within this FPS generator, I will make use of the update method, which is commonly used for processing elements, each time there is a frame. Within the update method is where I will update the years. I plan to use a counter variable (increases to a specified point before processing, delaying time), which could be changed to increase or decrease the amount of time between years. Each time the update method is called, the program should wait the desired amount of time (by default 10 seconds, as mentioned above), increasing the year as well as carrying out other processing that needs to be done.

Immediately this method proved to have some significant problems. Having the program count to a specified number, is not only processing intensive, but also depends entirely on system performance. This means that time cannot be controlled across different systems.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are several icons: a green circular arrow, a red square, a grey camera, a grey square with a circle, a grey square with a diagonal line, and a colon. Below these icons, the terminal displays a list of 15 lines, each consisting of a small icon followed by the text "Years: <year>, Time Elapsed (seconds): <time>". The years listed are 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, and 78. The times listed are 0.8502305, 0.9469656, 1.6312046, 0.9001026, 0.8421778, 0.8488997, 0.8429959, 0.9394837, 1.0108017, 0.9481396, and 0.8497769 respectively. The icons likely represent different file types or terminal functions.

For example, through testing, a computer with a 5.4Ghz processor was able to update the year on average every 0.9 seconds (see left). Fluctuations, while expected, are in this example great, and on a greater scale can cause inefficiencies, such as the movement and stats of an organism being out of sync. The fluctuations increase the lower the system performance.

For comparison, a computer with a 1.7Ghz processor was only able to update on average every 1.4 seconds (see below). Furthermore, other background processes significantly held back the program, with the 1.7Ghz processing updating the year every 3 seconds occasionally. This difference between these two highlights that this method would not produce the ability to have a constant time span.

```
Years: 7, Time Elapsed (seconds): 1.3239098
Years: 8, Time Elapsed (seconds): 2.7716897
Years: 9, Time Elapsed (seconds): 3.08004
Years: 10, Time Elapsed (seconds): 3.1629677
Years: 11, Time Elapsed (seconds): 2.4739618
Years: 12, Time Elapsed (seconds): 1.4505014
Years: 13, Time Elapsed (seconds): 1.3293929
Years: 14, Time Elapsed (seconds): 1.5027132
```

This method also presented further problems, for example the test here was with the program counting to 2,000,000,000, which took between the two PC's an average of 1 second. Making this extend to 10 seconds as planned would not be possible within Java, as the largest comparable number is 2,147,483,647 (32 bits). The final major error with this method was the fact that the user would be unable to accurately change time, as each computer is different, each change would have a different effect.

See below for the code used to test this method.

```
1.  public void update() {
2.      long currentFrame = 0; // counting the amount of times to run for
3.      int years = 0; // counting years
4.      long lastYearTime = System.nanoTime(); // the time at the start
5.
6.      while (gameRunning) { // loop to continuously check
7.          currentFrame++;
8.
9.          if (currentFrame == 2000000000) { // Not trackable, depends on system performance
10.             currentFrame = 0; // resetting current frame
11.             years++; // increasing years
12.
13.             long currentTime = System.nanoTime();
14.             long timeElapsedNanos = currentTime - lastYearTime;
15.             double timeElapsedSeconds = (double) timeElapsedNanos / 1e9; // Convert
nanoseconds to seconds
16.
17.             System.out.println("Years: " + years + ", Time Elapsed (seconds): " +
timeElapsedSeconds);
18.
19.             lastYearTime = currentTime; // Update the last year time
20.             gameRunning = false;
21.         }
22.     }
23. }
```

Java

3.2.1.1 Changing game speed

Considering the above factors, I have instead chosen to take advantage of Java's Runnable, which is already implemented within the FPS counter. Runnable allows the program to stop updating for defined periods of time, no matter the system performance. This is a much more reliable method of allowing the program to wait, and can also be changed more reliably, across the entire project. The program will update organisms and increase the year, before waiting 10 seconds to repeat the process. Runnable's sleep function pauses the processing for a defined amount of time, in this case 10 seconds is 10000.

```

1.     public void run() {
2.         double drawInterval = (double) 1000000000 / FPS; // Time interval between desired
frames
3.         double delta = 0;
4.         long lastTime = System.nanoTime();
5.         long currentTime;
6.         long timer = 0;
7.         long drawCount = 0;
8.
9.         while (thread != null) {
10.             currentTime = System.nanoTime();
11.             delta += (currentTime - lastTime) / drawInterval; // Time between last frame and
delta
12.             timer += (currentTime - lastTime); // Time since last frame
13.             lastTime = currentTime;
14.             if(delta >= 1){ // repainting and updating
15.                 try {
16.                     update();
17.                     years++;
18.                     Thread.sleep(timeFactor); // 1000 is every second, 10000 is every 10
seconds
19.                 } catch (InterruptedException e) {
20.                     throw new RuntimeException(e);
21.                 }
22.                 repaint();
23.                 delta--;
24.                 drawCount++;
25.             }
26.             if(timer >= 1000000000) { // Checks to see if a second has passed
27.                 currentFPS = drawCount; // how many frames during the previous second
28.                 drawCount = 0;
29.                 timer = 0;
30.             }
31.         }
32.     }

```

Java

As seen within the try catch method, the program does exactly that, updating the year every 10 seconds, as shown below.

```

Year: 1 Update duration (seconds): 10.0040557
Year: 2 Update duration (seconds): 10.0098195
Year: 3 Update duration (seconds): 10.0018786
Year: 4 Update duration (seconds): 10.0122232
Year: 5 Update duration (seconds): 10.0127159

```

This is significantly more accurate each time and does not depend on system performance. This means that the time for years can be the same across devices, keeping evolution synced and constant. Changes to the timeFactor variable is also consistent, adding a button to the screen, and a listener to halve the timeFactor when clicked, also shows to work. Once the button is clicked, the program will continue to the original factor, but on the next iteration, will run to the new factor. If the original factor was 10000 (10 seconds), and it was halved once, then the new factor is 5000. See below for the test.

```

Year: 1 Update duration (seconds): 10.0040557
Year: 2 Update duration (seconds): 10.0098195
Year: 3 Update duration (seconds): 10.0018786
Year: 4 Update duration (seconds): 10.0122232
Year: 5 Update duration (seconds): 10.0127159
5000
Year: 6 Update duration (seconds): 10.0094884
Year: 7 Update duration (seconds): 5.001069
Year: 8 Update duration (seconds): 5.0118051
Year: 9 Update duration (seconds): 5.0132373
Year: 10 Update duration (seconds): 5.0076323

```

Time is now not only present within EvSim, but changeable by the user, and can be used to update the organisms.

3.2.1.2 Pausing the game

The second feature related time, is the ability to pause the simulation, it is fundamental to allow the user to conduct hypotheses, as well as for testing. If the simulation was to run consistently, it would give a time frame for which testing can undergo, and this therefore is a weakness of the program. To keep track of whether the game is paused, the game window class will contain a gamePaused boolean variable, which when paused will stop the organisms from moving and evolving. The program should however still have the ability for interaction with organisms and menus.

To add logic, the gamePaused boolean will be checked before updating the organisms and increasing the year. This means that the rest of the game can continue its operation while the organisms and time are paused.

To test this logic, a pause button will be added to the screen, which when pressed will either pause or play the game. The creation and implementation of a pause/play button on the right-hand side of the screen is shown below.

```
1. pauseGame = new JButton("Pause/Play"); // Creating and titling the button
2. pauseGame.addMouseListener(this); // Adding a listener for a mouse click
3. frame.add(pauseGame, BorderLayout.EAST); // Adding it to the right-hand side of the frame
```

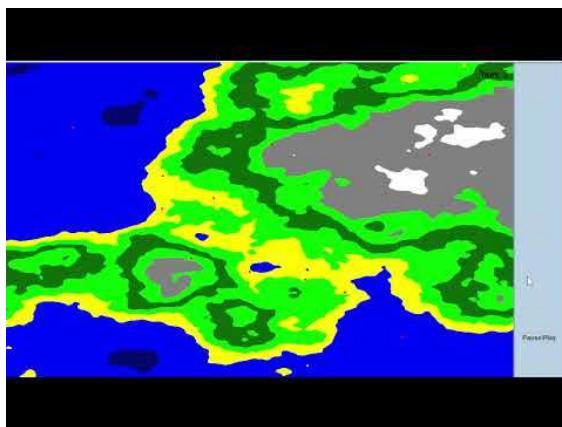
Java

The mouse listener for this is straightforward, with the boolean only needing to be flipped when the button is clicked. See below for the logic.

```
1. if(mouseEvent.getSource() == pauseGame) gamePaused = !gamePaused;
```

Java

In the video below, the pausing of the program is tested, with the display of years in the top right showing that time does not increase.



3.2.2 Organism Movement

3.2.1 Implementing the code from 2.3 Algorithm Design

Using the code from 2.3, more specifically 2.3.3 and 2.3.2.3, the organisms should move frequently, in small amounts, in directions that benefit themselves. With the organisms planned to be controlled through AI, the organism should be able to factor their current terrain and make a decision based off of the attributes within that terrain. Without the attributes currently available for each terrain block, the organisms will move based off the biome it is in, and then later in development, the organism can process more information. This should fit directly with the pseudocode designed in 2.3.3, more specifically the makeDecision method, with the program able to take only the current terrain and move a maximum of 20 pixels each time.

The method created in 2.3.3 should be sufficient in giving the organisms some base of movement, which can be expanded late in development. The main factors that will change between the two code blocks, is how JSON is read, and written to. Using the already created Organism class, an array of organisms can be created, directly mirroring what is in the JSON file, with each value changeable. I will also use the Jackson library here, with the same form as 3.1.3, allowing for the easy merge into the Organism class. Below is the Java equivalent of the pseudocode created in 2.3.3.

```

1.     private void makeDecision() {
2.         // Simplified decision-making logic, to be expanded
3.         if (!Objects.equals(currentTerrain, preferredTerrain)) {
4.             float randomScore = random.nextInt(0, 1);
5.             // Uses random score to chance the organism making a large or a small move
6.             if (randomScore > 0.5) {
7.                 // Make a larger move to a new location
8.                 // Can only move 20 pixels in one direction maximum
9.                 currentX += random.nextInt(-20, 20);
10.                currentY += random.nextInt(-20, 20);
11.            } else {
12.                // Make a smaller move to a new location
13.                // Can only move 10 pixels in one direction maximum
14.                currentX = currentX + random.nextInt(-10, 10);
15.                currentY = currentY + random.nextInt(-10, 10);
16.            }
17.        } else {
18.            // Smaller move to stay within the same area
19.            // Can only move 1 tile in one direction maximum
20.            currentX = currentX + random.nextInt(-5, 5);
21.            currentY = currentY + random.nextInt(-5, 5);
22.        }
23.    }

```

Java

Validation for the new values (current and currentY) is not yet in place. It is needed to ensure the values are below that of the game window, if they are not, it returns an out of bounds error. As mentioned within the comments, the organism movement will be expanded later in development, to take in more values and do more complex calculations deciding movement.

3.2.2 Fixing and updating organism movement

Validating the new coordinates was simple, with the new coordinates being compared to the size of the game window, and if larger or smaller than 0, then it would be adjusted the corresponding way by 10. Adjusting the position by 10 instead of setting it to the game window size, means there is a greater chance for the organism to move away from the border, therefore having less collisions.

```

1.         // This ensures that the new X and Y coordinates are not outside the bounds of the
window
2.         // The x and y coordinate are compared against the width and height of the gameWindow,
if higher or lower,
3.         // the coordinates are adjusted in the appropriate direction
4.         if (currentX >= GameWindow.WIDTH) {
5.             currentX = GameWindow.WIDTH - 10;
6.         } else if (currentX < 0) currentX = 10;
7.         if (currentY >= GameWindow.HEIGHT) {
8.             currentY = GameWindow.HEIGHT - 10;
9.         } else if (currentY < 0) currentY = 10;

```

Java

The code above will be inserted on line 23 of the makeDecision() method.

As mentioned previously, the decision method uses variables that are defined outside of the method, with the method itself also being private. The decision method is called within the constructor of the organisms' class, which is where variables are also defined. Using Jackson in a similar method to 3.1.3.3, an array of organism classes is created. When the array is iterated through, each organism's stats, current coordinates and terrain can be fetched and set globally, before calling the stats and movement

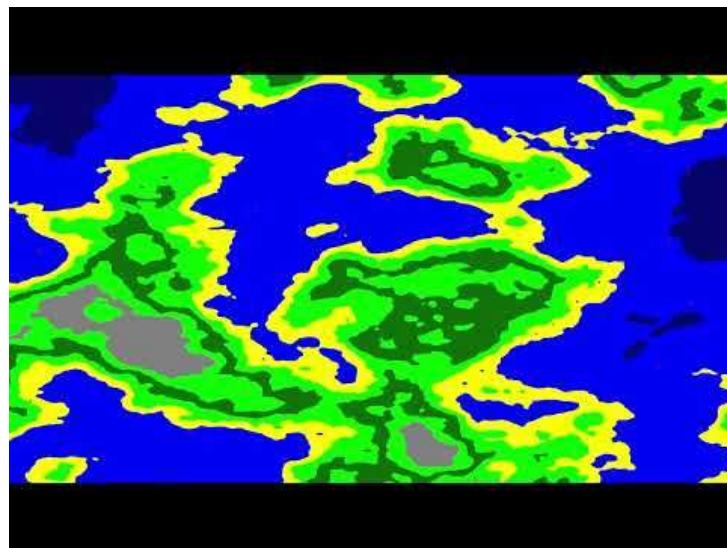
to be updated. Once the methods have been called, the old data is replaced, and saved to the JSON file. See below for the organisms' constructor method, which currently updates the coordinates of an organism each time the organisms are drawn to the screen (see 3.2.1).

```

1.  private String preferredTerrain;
2.  private String currentTerrain;
3.  private int currentX;
4.  private int currentY;
5.  private final Random random = new Random();
6.  public organisms() {
7.      ObjectMapper objectMapper = new ObjectMapper();
8.      try {
9.          Organism[] organisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
10.         for (Organism organism : organisms) {
11.             // setting the current attributes for the organism
12.             currentTerrain = organism.getCurrentTerrain();
13.             currentX = organism.getxCoordinate();
14.             currentY = organism.getyCoordinate();
15.             this.preferredTerrain = organism.getPreferredTerrain();
16.
17.             // Calling the methods which update the organisms position
18.             makeDecision();
19.
20.             organism.setCurrentTerrain(generateMap.getTerrain(currentX, currentY)); // Saves the new information from makeDecision
21.             organism.setxCoordinate(currentX);
22.             organism.setyCoordinate(currentY);
23.         }
24.         objectMapper.writeValue(new File("organisms.json"), organisms);
25.     } catch (Exception e) {
26.         e.printStackTrace();
27.     }
28. }
```

Java

This correctly updates the JSON, and as expected, changes the organisms on screen at the same time. See the video below for demonstration of this.



The organisms within this video are represented by red dots, if necessary, enlarge the video add visibility.

This demonstrates that the organisms move around the map sufficiently, and do not stay in one area necessarily all the time, following the code, the organism makes larger moves when it is not in its preferred terrain, and smaller moves when it is. There is also a 50% chance for the organism to make a large move whether it is in the preferred terrain or not.

3.2.3 Organism Evolution

Once again this will follow closely to the layout designed in 2.3.3, with each organism's stats changing depending on the biome around it. This will be reliant on 2.5.4 Terrain Attributes to also be designed. See below.

3.2.3.1 Terrain Attributes

To contain the terrain attributes classes, I have created a package file within the source files for EvSim, with each terrain having its own class, which inherits from a base class, TerrainAttributes. This is similar to the structure of 2.5.4, where each biome inherits base variables from a baseAttributes class, however in this one the terrains are contained in different files as opposed to one file for all. Below is the default terrain attributes class which all terrains will inherit from.

```
1. package Terrains;
2. import java.util.Random;
3. public class TerrainAttributes {
4.     final Random random = new Random();
5.     public String mainFoodSource = "Plants";
6.     public double healthGain = 0.125;
7.     public double speedGain = 0.5;
8.     public int temperature = 20;
9.     public double oxygenAvailability = 100;
10.    public int foodAvailability = 50;
11.    public int waterAvailability = 50;
12.    public int terrainId = 0;
13. }
```

Java

As within design, the values in here are set as a balance between all terrains, however each one is very likely to be changed or unused across terrains. Below is the grass terrain, extending terrain attributes.

```
1. package Terrains;
2. public class Grass extends TerrainAttributes {
3.     public Grass(){
4.         healthGain = healthGain + 0.2;
5.         speedGain = speedGain + 1.5;
6.         temperature = temperature - random.nextInt(-5,10);
7.         foodAvailability = 100;
8.         waterAvailability = 50;
9.         terrainId = 1;
10.    }
11. }
```

Java

This demonstrates how each terrain can use the default attributes, only making changes where needed. This significantly reduces the amount of reused code across the program. The grass terrain can also act as a parent class to other terrains, for example the forest class, seen below.

```
1. package Terrains;
2. public class Forest extends Grass {
3.     public Forest() {
4.         healthGain = healthGain + 0.4;
5.         speedGain = speedGain - 2.5;
6.         foodAvailability = 100;
7.         waterAvailability = 100;
8.         terrainId = 6;
9.     }
10. }
```

Java

These values are unlikely to be correct, however they cannot be tested and configured until the organisms themselves act on these. 3.2.3.1's intention was to have the terrain attributes defined however, even if they are not yet perfect. See 3.2.3.3 for testing and alterations.

3.2.3.2 Updating the organisms' stats

This will once again follow along with design, to be expanded later. Using the organisms class, similar to the organism movement, the organisms stats can be fetched, before having the calculations performed, and then updated again within the JSON. This will rely heavily on the Stats class created in 3.1.3.3, as each organism's stats are sent to this class.

The first part of this section requires the terrain attributes (3.2.3.1) to be fetched, into an object that contains the corresponding variables. For this a getTerrainAttributes function will be created, with a switch case comparing the current terrain to the available terrains, returning the correct class.

```

1.  private TerrainAttributes getTerrainAttributes() {
2.      TerrainAttributes terrainAttributes;
3.
4.      terrainAttributes = switch (currentTerrain) {
5.          case "Grass" -> new Grass(); // Returns the grass class from Terrains package
6.          case "Water" -> new Water(); // Returns the water class from Terrains package
7.          case "Mountain" -> new Mountain();
8.          case "Beach" -> new Beach();
9.          case "Snow" -> new Snow();
10.         case "Forest" -> new Forest();
11.         case "Deep Water" -> new DeepWater();
12.         default -> new TerrainAttributes();
13.     };
14.     return terrainAttributes;
15. }
```

Java

To test this, a print statement was added displaying the current terrain id, as well as the organisms current terrain name. Below is the console output, demonstrating the correct return of classes.

```

Year: 38 Update duration (seconds): 0.6286932
Terrain Id: 4 Current Terrain: Beach
Terrain Id: 2 Current Terrain: Water
Year: 39 Update duration (seconds): 0.6282941
Terrain Id: 2 Current Terrain: Water
Terrain Id: 2 Current Terrain: Water
Year: 40 Update duration (seconds): 0.6276312
Terrain Id: 4 Current Terrain: Beach
Terrain Id: 2 Current Terrain: Water
```

The organisms can now be updated using the same logic as presented in 2.3.3, logic which will need to be changed and expanded later in development to increase the scale and accuracy of evolution. As mentioned earlier this will be reliant on 3.1.3.3, to access and change the organisms current statistics.

```

1. private void updateStats() {
2.     TerrainAttributes terrainAttributes = getTerrainAttributes();
3.
4.     if (stats.getAge() == 0){
5.         stats.setPreferredFood(terrainAttributes.mainFoodSource); // if the organism is
new, then the null attributes are set
6.         stats.setPreferredTemperature(terrainAttributes.temperature);
7.     }
8.
9.     if (stats.getHealth() < 100) {
10.         stats.setHealth(stats.getHealth() + terrainAttributes.healthGain); // if the
organism is not on 100% health, then their health will slowly increase
11.     }
12.
13.     if (stats.getAge() % 5 == 0){
14.         stats.setSpeed(stats.getSpeed() + terrainAttributes.speedGain); // every 5 years
the speed of the organism increases
15.     }
16.
17.     if (terrainAttributes.temperature > stats.getPreferredTemperature() + 5) { // If the
temperature of the current biome is different to the preferred temperature, then the health and the
preferred temperature change
18.         stats.setHealth(stats.getHealth() - 0.3);
19.         stats.setPreferredTemperature(stats.getPreferredTemperature() + 1);
20.     } else if (terrainAttributes.temperature < stats.getPreferredTemperature() - 5) {
21.         stats.setHealth(stats.getHealth() - 0.3);
22.         stats.setPreferredTemperature(stats.getPreferredTemperature() - 1);
23.     } else if (terrainAttributes.temperature > stats.getPreferredTemperature() + 10) {
24.         stats.setHealth(stats.getHealth() - 0.75);
25.         stats.setPreferredTemperature(stats.getPreferredTemperature() + 2);
```

99

```

26.        } else if (terrainAttributes.temperature < stats.getPreferredTemperature() - 10) {
27.            stats.setHealth(stats.getHealth() - 0.75);
28.            stats.setPreferredTemperature(stats.getPreferredTemperature() - 2);
29.        } else {
30.            stats.setHealth(stats.getHealth() - random.nextDouble(-0.1, 0.1));
31.            stats.setPreferredTemperature(stats.getPreferredTemperature() - random.nextInt(-1,
1));
32.        }
33.
34.        if (!Objects.equals(terrainAttributes.mainFoodSource, stats.getPreferredFood())) {
35.            stats.setHealth(stats.getHealth() - random.nextInt(0, 5)); // different food type
means the organism health decreases
36.        }
37.        if (random.nextInt(0, 100) < 5) {
38.            stats.setPreferredFood(terrainAttributes.mainFoodSource); // small chance for
preferred food source to change alongside the biome
39.        }
40.
41.        if (terrainAttributes.foodAvailability < 20) { // based on the amount of food that is
available to the organism, their retention and health increases/decreases
42.            stats.setHealth(stats.getHealth() - random.nextInt(2, 10));
43.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(2, 10));
44.        } else if (terrainAttributes.foodAvailability < 40) {
45.            stats.setHealth(stats.getHealth() - random.nextInt(1, 6));
46.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(2, 10));
47.        } else if (terrainAttributes.foodAvailability < 60) {
48.            stats.setHealth(stats.getHealth() - random.nextInt(0, 3));
49.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(0, 3));
50.        } else if (terrainAttributes.foodAvailability < 80) {
51.            stats.setHealth(stats.getHealth() - random.nextInt(-2, 2));
52.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(-2, 2));
53.        } else stats.setRetentionRate(stats.getRetentionRate() - random.nextInt(0, 5)); // Food
is surplus, retention rate decreases
54.
55.        if (terrainAttributes.waterAvailability < 20) { // based on the amount of water that is
available to the organism, their retention and health increases/decreases
56.            stats.setHealth(stats.getHealth() - random.nextInt(2, 10));
57.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(2, 10));
58.        } else if (terrainAttributes.waterAvailability < 40) {
59.            stats.setHealth(stats.getHealth() - random.nextInt(1, 6));
60.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(2, 10));
61.        } else if (terrainAttributes.waterAvailability < 60) {
62.            stats.setHealth(stats.getHealth() - random.nextInt(0, 3));
63.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(0, 3));
64.        } else if (terrainAttributes.waterAvailability < 80) {
65.            stats.setHealth(stats.getHealth() - random.nextInt(-2, 2));
66.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(-2, 2));
67.        } else stats.setRetentionRate(stats.getRetentionRate() - random.nextInt(0, 5)); // Food
is surplus, retention rate decreases
68.
69.        if (terrainAttributes.oxygenAvailability < 50) { // if oxygen is scarce, harsh hit on
health, but gain in retention rate
70.            stats.setHealth(stats.getHealth() - random.nextInt(0, 10));
71.            stats.setRetentionRate(stats.getRetentionRate() + random.nextInt(0, 10));
72.        } else stats.setHealth(stats.getHealth() + random.nextInt(1, 5));
73.
74.        // if the organism has a high retention rate, they are able to negate the effects of
reduced oxygen/water/food
75.        if (stats.getRetentionRate() > 80) stats.setHealth(stats.getHealth() +
random.nextInt(4, 15));
76.        else if (stats.getRetentionRate() > 60) stats.setHealth(stats.getHealth() +
random.nextInt(3, 12));
77.        else if (stats.getRetentionRate() > 40) stats.setHealth(stats.getHealth() +
random.nextInt(2, 10));
78.        else if (stats.getRetentionRate() > 20) stats.setHealth(stats.getHealth() +
random.nextInt(1, 5));
79.        else stats.setHealth(stats.getHealth() + random.nextInt(0, 1));
80.
81.        if (stats.getHealth() > 100) {
82.            stats.setHealth(100);
83.        }

```

```

84.         stats.setAge(stats.getAge() + 1); // adding 1 year to the organisms age
85.
86.     }

```

Java

The Java equivalent is very similar to 2.3.3, with only the use of stats and terrain attributes different, this is mainly due to much of this section being if/else statements. As mentioned, this will be expanded and changed to increase accuracy of evolution, and to include more of the stats. Currently, attributes such as awareness, attack and defence are unused throughout the program, highlighting the need for expansion of this class. See 3.3.

3.2.4 Displaying stats to the user

While this section is not specifically calculations and processing, it is the link between the calculations and front end, therefore a part of visually representing evolution to the user. This section will cover the creation of the menu once the user is in the simulation. This menu will contain a display of the overall organism statistics, such as organism count, most common terrain, fastest organism, healthiest organism etc. The menu will also allow the user to pause, speed up and slow down time. Within this section logic will be implemented which allows the user to select an organism to review its specific statistics, which will replace the overall view when an organism is selected.

3.2.4.1 Design of the menu

Once again I will use the IntelliJ forms creator to design and add in the side menu to the simulation's window. The forms creator significantly reduces the need for tedious layouts of UI such as JButtons and JLabels, which would take up time that could otherwise be spent on evolution expansion. On the right, the forms layout shows what is available to the user during the simulation. Many of these features will have to be updated consistently during the running of the simulation, for example the JLabels are blank at the moment (see Current Year and Current Game Speed). Buttons will also have to have added functionality, some which already have logic, for example the Pause Game button and Slow Down and Speed Up.

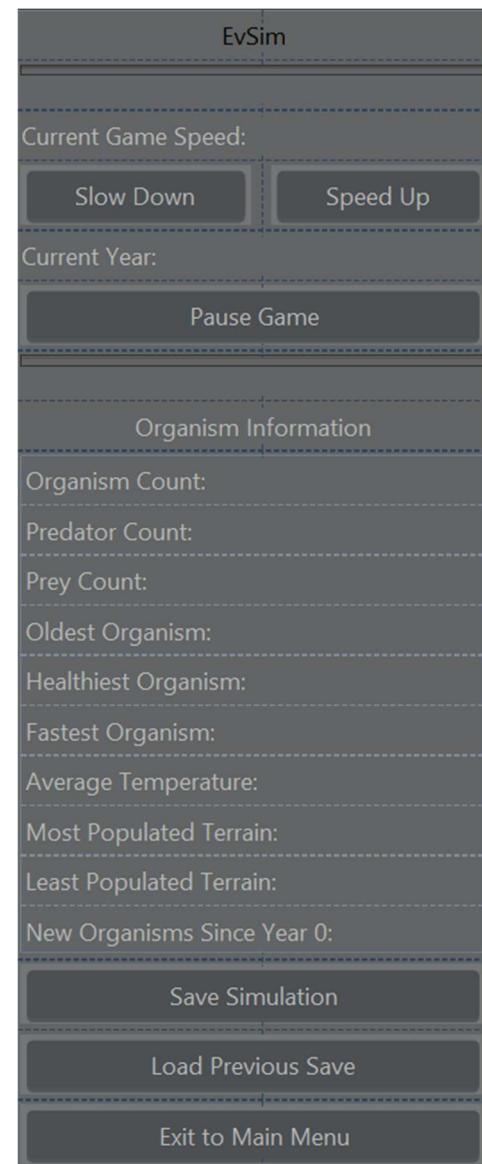
The forms creator only shows a basic class once the form has been designed, with private attributes for buttons and panels. The panel SideMenu contains all of the features such as buttons and labels, meaning only that panel needs to be added into the game window. Due to the panel being private and not accessible to the form if static, a function which returns the panel will allow this panel to be added to the game window. See below.

```

1. public class SideMenu {
2.     private JPanel SideMenu;
3.     public JPanel getSideMenu(){
4.         return SideMenu;
5.     }
6. }

```

Java



Using the border layout, the menu can be added to the west side of the window, giving the user an overview of the global statistics.

3.2.4.2 Adding functionality to the side menu

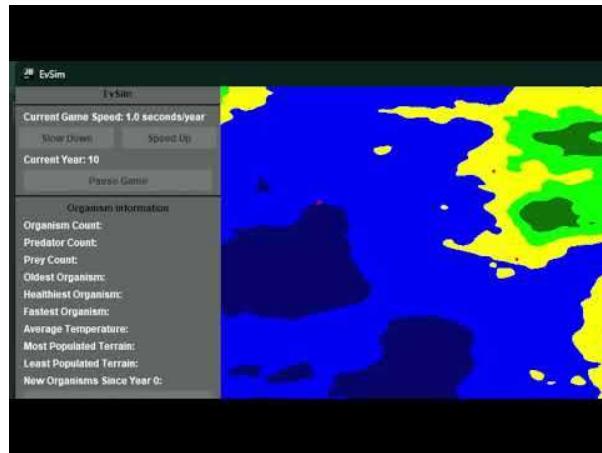
I will start by adding functionality to the buttons, and then move onto updating the labels. As previously mentioned, some of the buttons already have logic, it is only the transfer code which is required. See 3.2.1.2 for the implementation of time and the ability to pause.

The option for the user to save and load a game will stay inactive for the time being, as there is not yet the development of this feature. This will become available later in development. The buttons will react to a mouse click, as with other buttons across EvSim. This section should also ensure the Current Game Speed label is also updated, which only requires the text to be set. The option to slow down, speed up and pause are shown in code below.

```
1.     public void mouseClicked(MouseEvent mouseEvent) {
2.         // Every 1000 is a second, anything longer than 8.5 seconds will be too long.
3.         // Anything below 0.5 seconds is too short.
4.         if(mouseEvent.getSource() == slowDownButton && GameWindow.timeFactor < 8500)
GameWindow.timeFactor = GameWindow.timeFactor + 500; // Increase the seconds per year by 0.5
seconds
5.         if(mouseEvent.getSource() == speedUpButton && GameWindow.timeFactor > 500)
GameWindow.timeFactor = GameWindow.timeFactor - 500; // Decrease the seconds per year by 0.5
seconds
6.         if(mouseEvent.getSource() == pauseGameButton) {
7.             GameWindow.gamePaused = !GameWindow.gamePaused; // Flip the paused boolean
8.             if(Objects.equals(pauseGameButton.getText(), "Play Game"))
pauseGameButton.setText("Pause Game");
9.             else pauseGameButton.setText("Play Game"); // Flip the text shown to equate whether
the game is paused or not
10.        }
11.        // For every 1000 in gameWindow.timeFactor, it is 1 second. Therefore, to divide it by
1000 is the time in seconds.
12.        currentGameSpeedLabel.setText("Current Game Speed: " + (double) GameWindow.timeFactor /
1000 + " seconds/year"); // Update the label to show the current game speed in seconds per year
13.    }
```

Java

The current game speed should be the only label to be updated within mouseClicked, as it is only changed when the mouse is clicked on one of the appropriate buttons, therefore does not need to be constantly updated, as opposed to the current year for example. See below for a demonstration of these 4 sections working.



For the other labels, such as the year, and all those underneath organism information, this will have to be updated each year. For this I have created an UpdateUI method, which will be called each time a year passes, similar to how the organisms are updated. Within this, each label will have its text updated to display the correct information, for example the years will be updated the following way:

```
1. currentYearLabel.setText("Current Year: " + GameWindow.years);
```

See below for the full method, which updates all the labels.

```

1.    // This function updates the UI to show the most up-to-date statistics globally.
2.    // Takes the organisms' statistics from JSON, and updates for the relevant label
3.    // Refer to comments for explanation of specific sections
4.    public void UpdateUI(){ // Called each time the game window paints to the screen
5.        // Defining the attributes that are to be used
6.        int oldestOrganism = 0, healthiestOrganism = 0, fastestOrganism = 0, averageTemp;
7.        String mostCommonTerrain = "N/A", leastCommonTerrain = "N/A";
8.        ObjectMapper objectMapper = new ObjectMapper();
9.        try {
10.            Organism[] organisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
11.
12.            // The length is equivalent to the number of objects and therefore the number of
organisms
13.            // Once organisms have the ability to die, this will have to update to only count
alive organisms
14.            organismCountLabel.setText("Organism Count: " + organisms.length);
15.
16.            // Temporary variables which are used to compare organisms
17.            int tempAge = organisms[0].getStats().getAge();
18.            double tempHealth = organisms[0].getStats().getHealth();
19.            double tempSpeed = organisms[0].getStats().getSpeed();
20.            double totalTemp = 0;
21.            int grassCount = 0, waterCount = 0, mountainCount = 0, beachCount = 0, snowCount =
0, forestCount = 0, deepWaterCount = 0;
22.
23.            // Loops through each organism and compares statistics
24.            // Refer to comments for explanation of specific sections
25.            for (Organism organism : organisms) {
26.                // Finds the organism with the highest age
27.                if (tempAge < organism.getStats().getAge()) {
28.                    oldestOrganism = organism.getOrganismId();
29.                    tempAge = organism.getStats().getAge();
30.                }
31.                // Finds the organism with the highest health
32.                if (tempHealth < organism.getStats().getHealth()) {
33.                    healthiestOrganism = organism.getOrganismId();
34.                    tempHealth = organism.getStats().getHealth();
35.                }
36.                // Finds the organism with the highest speed
37.                if (tempSpeed < organism.getStats().getSpeed()) {
38.                    fastestOrganism = organism.getOrganismId();
39.                    tempSpeed = organism.getStats().getSpeed();
40.                }
41.
42.                // Adds the preferred temperature of each organism
43.                // Is used to get the mean temperature
44.                totalTemp = totalTemp + organisms[0].getStats().getPreferredTemperature();
45.
46.                // To find how many organisms are in each terrain
47.                switch (organism.getCurrentTerrain()) {
48.                    case "Grass" -> grassCount++;
49.                    case "Water" -> waterCount++;
50.                    case "Mountain" -> mountainCount++;
51.                    case "Beach" -> beachCount++;
52.                    case "Snow" -> snowCount++;
53.                    case "Forest" -> forestCount++;
54.                    case "Deep Water" -> deepWaterCount++;
55.                }
56.            }
57.            // Variables which are used to compare how common terrains are, as well as set the
equivalent terrain name
58.            int[] terrainCounts = {grassCount, waterCount, mountainCount, beachCount,
snowCount, forestCount, deepWaterCount};
59.            String[] terrainNames = {"Grass", "Water", "Mountain", "Beach", "Snow", "Forest",
"Deep Water"};
60.            int maxCount = 0, minCount = Integer.MAX_VALUE; // Set to MAX_VALUE so that there
can be a large amount of organisms in a terrain, will never reach this value
61.
62.            // Checks for both the most common and least common terrain
63.            // Refer to comments for explanation of specific sections

```

```

64.             for (int i = 0; i < terrainCounts.length; i++) {
65.                 if (terrainCounts[i] > maxCount) { // If there are more organisms than the
66.                     previous
67.                     maxCount = terrainCounts[i]; // Current highest organism count
68.                     mostCommonTerrain = terrainNames[i]; // Sets the terrain name as the most
69.                     common
70.                 }
71.                 if (terrainCounts[i] < minCount) { // If there are fewer organisms than the
72.                     previous
73.                     minCount = terrainCounts[i]; // Current lowest organism count
74.                     leastCommonTerrain = terrainNames[i]; // Sets the terrain name as the least
75.                     common
76.                 }
77.             }
78.             averageTemp = (int) (totalTemp / organisms.length); // Calculates the mean
79.             temperature (average)
80.         }
81.         // Updates each label, some have HTML format, to show the specific stat underneath
82.         // as a bullet point, refer to documentation 3.2.4.2
83.         currentYearLabel.setText("Current Year: " + GameWindow.years); // Displays the
84.         years held in GameWindow
85.         oldestOrganismLabel.setText("<html><p>Oldest Organism ID: " + oldestOrganism +
86.             "</p><ul><li>Age: " + organisms[oldestOrganism].getStats().getAge() + "</li></ul></html>");
87.         predatorCountLabel.setText("Predator Count: 0 (disabled)");
88.         preyCountLabel.setText("Prey Count: 0 (disabled)");
89.         healthiestOrganismLabel.setText("<html><p>Healthiest Organism ID: " +
90.             healthiestOrganism + "</p><ul><li>Health: " + organisms[healthiestOrganism].getStats().getHealth() +
91.             "+ "</li></ul></html>\"");
92.         fastestOrganismLabel.setText("<html><p>Fastest Organism ID: " + fastestOrganism +
93.             "</p><ul><li>Speed: " + organisms[fastestOrganism].getStats().getHealth() + "</li></ul></html>\"");
94.         averageTemperatureLabel.setText("Average Temperature: " + averageTemp);
95.         mostPopulatedTerrainLabel.setText("<html><p>Most Popular Terrain: " +
96.             mostCommonTerrain + "</p><ul><li>Count: " + maxCount + "</li></ul></html>\"");
97.         leastPopulatedTerrainLabel.setText("<html><p>Least Popular Terrain: " +
98.             leastCommonTerrain + "</p><ul><li>Count: " + minCount + "</li></ul></html>\"");
99.         newOrganismsSinceYearLabel.setText("New Organisms Since Year 0: N/A");
100.     } catch (IOException e) {
101.         e.printStackTrace();
102.     }
103. }

```

Java

Refer to comments throughout this code block, as these explain each section of the code. Some labels are updated with html format, this is to add in both the ID of the organism, as well as the actual statistic, for example the organism with the highest health is organism 34, with 79.8. Below is a visual representation of what this updated side menu looks like during the simulation.

This will allow for much easier testing of organism statistics, as errors can be seen here, with organism 16 having a speed of 100. This, combined with the next section (3.2.4.3), will also allow testing throughout the simulation. 3.2.4.3 covers the ability for the user to click on an organism, or find an organism, with that organisms' statistics being viewed on the side menu.

Once features such as breeding cycles, predator and prey cycles are implemented, I will return to this specific section to add information to the labels which currently show N/A or (disabled), namely Predator Count, Prey Count, and New Organisms Since Year 0.



The only section which now requires functionality is the Exit to Main Menu button, however this will be linked to save logic, so I will leave it until that is available. See later in development for this functionality.

3.2.4.2 Specific organism statistics

This section covers the ability to find and/or click on an organism on screen to show its specific statistics.

The first task here was to recognise a mouse click, anywhere within the map, then fetch the X and Y coordinates of the mouse, comparing it to the current organisms to see if any are within that radius. The code below is what I am using to test this, with the X and Y already being a factor of the mouse listener class. The X coordinate of the mouse will have to be adjusted by -250 (as noted in the comments) due to the side menu taking 250px. Similarly, the y coordinate will also have to be adjusted by -30, as the top of the screen is offset by 30px.

```
1. public void mouseClicked(MouseEvent mouseEvent) {  
2.     System.out.println(mouseEvent.getX() - 250); // -250 to account for the menu  
3.     System.out.println(mouseEvent.getY() - 30); // -30 to account for top of screen  
4. }
```

Java

To test this, I will load a simulation with 2 organisms, pause the game, click on one of the organisms, and compare the x and y coordinates to what is contained in the JSON files. The coordinates of both will vary, as the user is unexpected to click on the exact coordinates of the organism, therefore when finding the specific organism within the code, bounds will be added of around 10 pixels. This will be enough to allow the user to click within a sufficient range but also to ensure the user doesn't click on the organism by mistake.

```
2 20  
3 557  
4 807  
5 432
```

Above is the console result after clicking on one of the organisms, comparing this to the JSON file, which is shown below.

```
1. [  
2.     {  
3.         "organismId": 0,  
4.         "preferredTerrain": "Beach",  
5.         "currentTerrain": "Beach",  
6.         "xCoordinate": 9,  
7.         "yCoordinate": 552,  
8.         "isDead": false  
9.     },  
10.    {  
11.        "organismId": 1,  
12.        "preferredTerrain": "Water",  
13.        "currentTerrain": "Water",  
14.        "xCoordinate": 797,  
15.        "yCoordinate": 429,  
16.        "isDead": false  
17.    }  
18. ]
```

JSON

*Stats omitted

Taking line 2 and 3 from the console, this is organism 0, and lines 4 and 5 being organism 2. Considering how close the mouse was to the organism conducting this test, and comparing this to the actual coordinates, a bound of 10 pixels as mentioned earlier will be more than sufficient, it may even be reduced in later testing. However, it appears that there are slight differences between the two coordinates. The mouse when clicked directly on the organism, however each coordinate is offset by ~10. To counteract this, I have increased the adjustments to be X - 260 and Y - 35.

Testing this once again, the results were much more accurate. See below.

```
2 1053  
3 158
```

```
1. [  
2.     {  
3.         "organismId": 0,  
4.         "preferredTerrain": "Forest",  
5.         "currentTerrain": "Forest",  
6.         "xCoordinate": 1054,  
7.         "yCoordinate": 161,  
8.         "isDead": false  
9.     },  
10.    {  
11.        "organismId": 1,  
12.        "preferredTerrain": "Water",  
13.        "currentTerrain": "Water",  
14.        "xCoordinate": 1314,  
15.        "yCoordinate": 568,  
16.        "isDead": false  
17.    }  
18. ]
```

Java

*Stats omitted

As seen from the highlighted section of code, organism 0 is at that location, with the mouse coordinates being much closer to the actual organism coordinates.

This next section covers how the specific organism which was clicked on is selected. For this, I will once again iterate through the JSON file, comparing the mouse coordinates to each organisms' coordinates. If the mouse clicks within the area of the organism, then it will output the organism ID, coordinates and mouse coordinates. See below for both code and test data.

```
1.     public void mouseClicked(MouseEvent mouseEvent) {  
2.         int newMouseX = mouseEvent.getX() - 260, newMouseY = mouseEvent.getY() - 35;  
3.  
4.         Organism[] organisms;  
5.         ObjectMapper objectMapper = new ObjectMapper();  
6.         try {  
7.             organisms = objectMapper.readValue(new File("organisms.json"), Organism[].class);  
// Creating an array of organisms from organisms.json  
8.             for (Organism organism : organisms) { // For each organism (object) in Organisms  
(array)  
9.                 int tempOrganismX = organism.getxCoordinate(), tempOrganismY =  
organism.getyCoordinate(); // Setting the current organisms coordinates into variables  
10.                int xDifference = Math.abs(newMouseX - tempOrganismX); // Returning the  
difference between mouse and organism coordinates  
11.                int yDifference = Math.abs(newMouseY - tempOrganismY); // Returns as an  
absolute number, so it can be compared to only one bound  
12.  
13.                    // If both are within the bound of 10  
14.                    if(xDifference <= 10 && yDifference <= 10) System.out.println("Organism ID: " +  
organism.getOrganismId() + " Organism X, Y: " + tempOrganismX + ", " + tempOrganismY + " Mouse X, Y  
: " + newMouseX + ", " + newMouseY);  
15.                }  
16.            } catch (IOException e) {  
17.                System.out.println(e);  
18.            }  
19.        }
```

Java

```
9 Organism ID: 0 Organism X, Y: 990, 514 Mouse X, Y : 990, 513  
10 Organism ID: 1 Organism X, Y: 1527, 700 Mouse X, Y : 1527, 699
```

As seen, when the mouse clicked within the range of an organism, then it returns the ID and coordinates as intended.

The next section will cover the display of information to the user.

For the specific organism information to be displayed to the user, the overall statistics will have to be hidden and replaced with the specific organism statistics. For the program to know when there is an organism selected, the GameWindow class will include an integer variable (organismSelected) holding the currently selected organism. If there is no organism selected, then the integer holds -1. SideMenu (3.2.4.1) can therefore check if there is an organism selected. If an organism is selected, then the stats are fetched from the index organismSelected within the array. See below for the implementation, as always refer to comments for explanation on specific sections.

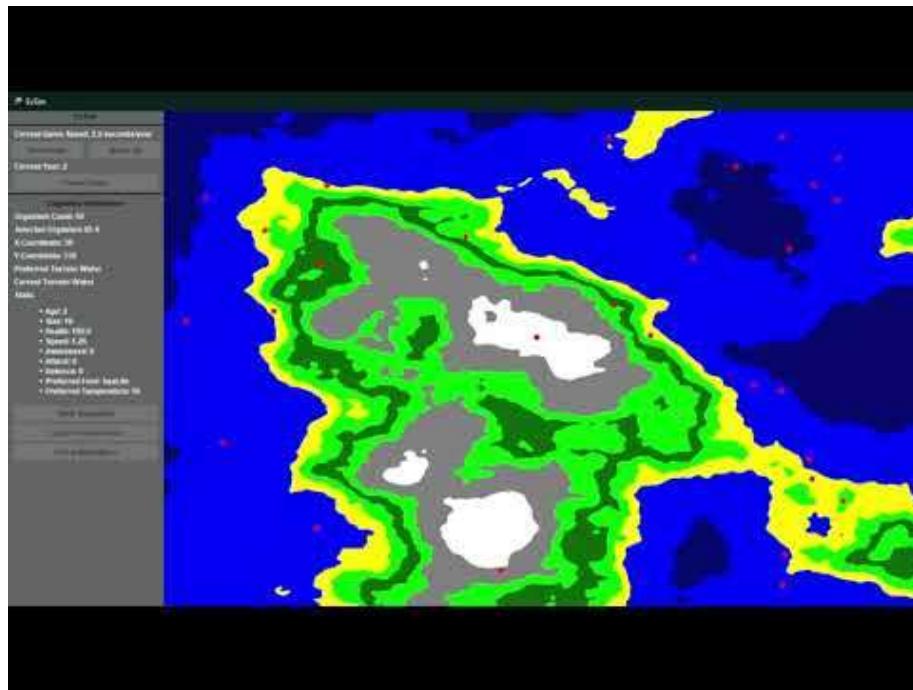
```

1.    // This function updates the UI to show either global or local organism statistics.
2.    // Takes the organisms' statistics from JSON, and updates for the relevant label
3.    // Refer to comments for explanation of specific sections
4.    public void UpdateUI(){ // Called each time the game window paints to the screen
5.        // Defining the attributes that are to be used
6.        int oldestOrganism = 0, healthiestOrganism = 0, fastestOrganism = 0, averageTemp;
7.        String mostCommonTerrain = "N/A", leastCommonTerrain = "N/A";
8.        ObjectMapper objectMapper = new ObjectMapper();
9.        try {
10.            Organism[] organisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
11.
12.            // If an organism is not selected, display global statistics
13.            if (GameWindow.organismSelected == -1) {
14.                See 3.2.4.2 [omitted as unchanged]
15.            } else {
16.                // Replace the labels with specific organism statistics
17.                predatorCountLabel.setText("Selected Organism ID: " +
organisms[GameWindow.organismSelected].getOrganismId());
18.                preyCountLabel.setText("X-Coordinate: " +
organisms[GameWindow.organismSelected].getxCoordinate());
19.                oldestOrganismLabel.setText("Y-Coordinate: " +
organisms[GameWindow.organismSelected].getyCoordinate());
20.                healthiestOrganismLabel.setText("Preferred Terrain: " +
organisms[GameWindow.organismSelected].getPreferredTerrain());
21.                fastestOrganismLabel.setText("Current Terrain: " +
organisms[GameWindow.organismSelected].getCurrentTerrain());
22.                // Displays the stats in HTML bullet point format
23.                averageTemperatureLabel.setText("<html><p>Stats: </p>" +
"

```

Java

Also see the video below for a video demonstrating how the side menu now displays the organism statistics, or global statistics if an organism is not selected. Note that the organisms are enhanced in size for the purpose of this video. Also make note that the statistics have not yet been calibrated and tested, so the statistics themselves may show wrong. See 3.3.



The side menu is now complete, correctly displaying and updating to show the desired information at the correct times.

3.3 Organism Statistics

This section focuses on the expansion and calibration of statistics. Currently, statistics are updated, however either update with odd values, or increase/decrease too fast. The aim of this section is to create logic and expand the organism statistics, which is the primary source of evolution within EvSim. This section will be broken down into specific attributes, see headings tab for selection. For each attribute, please refer to the comments within code blocks, as these will explain the specific sections as well as a larger overview. The write out will mainly focus on explaining the logic as opposed to explaining the code.

Each attribute will be calculated and updated through an individual method, for example `updateAwareness()`. Each of these methods will then be called within `updateStats`, linking them all together. JSON will be commonly referenced, with some additions made to how JSON is updated, for example organism history. Once again refer to specific comments for explanation of this and refer to 3.1.3 for the original JSON.

3.3.1 Awareness

3.3.1.1 Logic

The first addition made for awareness, was visibility within different terrains. For example, visibility on top of a mountain, compared to in a forest, is going to be different, and will affect how aware an organism can be. Each terrain now has a visibility attribute, listed below is the data for each.

Default -> 1.0, Grass -> 1.0, Water -> 0.6, Mountain -> 1.2, Beach -> 1.1, Snow -> 0.7, Forest -> 0.55, DeepWater -> 0.4

On the other hand, organisms will have a visibility radius within which they are able to see other organisms near them. The default value for this is a radius of 10px. This means that new organisms have

a very small area within which they can see other organisms. This factor will be affected by the terrain as well, for example visibility within grass will be 12px and visibility within deep water is 2px. This means the organisms in areas of extremely low visibility will have less chance of breeding and will also have less chance of being able to survive as predator/prey (once created).

To add in these features, JSON will be adjusted to include currentVisibility, previous visibility, as well as a count of organisms seen. For a new organism that is within the grass terrain for 3 years, which encounters 1 other organism, the data will be the following.

```

1. [
2.   {*
3.     "organismId": 0,
4.     "stats": {
5.       "currentAwareness": 1.16, **
6.       "organismsEncountered": 0.1,
7.       "awarenessHistory": [
8.         1.0,
9.         1.1,
10.        1.1
11.      ]
12.    },
13.    "preferredTerrain": "Grass",
14.    "currentTerrain": "Grass",
15.    "xCoordinate": 25,
16.    "yCoordinate": 653
17.  }
18. ]

```

JSON

*Other data omitted
**Decimal rounded to 2dp

Calculation for current awareness is shown below.

$$\text{awareness} = (\text{terrainVisibility} + \text{organismsEncountered})$$

$$\text{currentAwareness} = \frac{\text{awareness} + \text{previousAwareness}'}{\text{previousAwarenessCount} + 1} + \text{organismsEncountered}$$

The calculation should only consider a maximum of 5 previous years. See below for the calculations for each year.

Current Awareness Year 1 -> $((1.0 + 0) + 0 / 0 + 1) + 0 = 1.0$

Current Awareness Year 2 (organism encountered) -> $((1.0 + 0.1) + 1.0) / 1 + 1) + 0.1 = 1.15$

Current Awareness Year 3 (unchanged) -> $((1.0 + 0.1) + 1.1 + 1.0) / 2 + 1) + 0.1 = 1.16$

Note that awarenessHistory only stores the result of awareness.

With the example above only considering one encounter and one terrain over a span of 3 years, I will provide an example of a larger data set below.

In this example an organism has visited grass (years 1-10), forest (years 10-13) and grass again (years 13-20). The organism encountered 3 other organisms (years 5, 12, 15). See below for the resultant awareness after 20 years.

Year 1 -> 1.0	Year 6 -> 1.13	Year 11 -> 1.05	Year 16 -> 1.44
Year 2 -> 1.0	Year 7 -> 1.15	Year 12 -> 1.09	Year 17 -> 1.51
Year 3 -> 1.0	Year 8 -> 1.16	Year 13 -> 1.125	Year 18 -> 1.51
Year 4 -> 1.0	Year 9 -> 1.18	Year 14 -> 1.14	Year 19 -> 1.51
Year 5 -> 1.12	Year 10 -> 1.12	Year 15 -> 1.35	Year 20 -> 1.51

As shown, the awareness is balanced, dependant on the terrain as well as the organisms encountered. This provides a learning curve for the organisms, being more aware the more organisms met.

3.3.1.2 First Implementation

The updating of awareness within the project code is split into 3 sections, the main updateStats method, where all stats will be updated, the updateAwareness method, which calculates and makes appropriate changes, and checkOrganismProximity, which updates the organisms encountered variable.

Update stats is simple and will be the same for all statistics, only calling the update* method. * Being the name of the statistic. In this case it is as follows:

```
1. private void updateStats() {
2.     TerrainAttributes terrainAttributes = getTerrainAttributes();
3.
4.     updateAwareness(terrainAttributes); // Updating the organism's awareness
5.
6.     stats.setAge(stats.getAge() + 1); // adding 1 year to the organisms age
7. }
```

Java

The updateAwareness method follows the same logic as 3.3.1.1, with the calculations implemented and set to update the JSON after. The code for this method is below, as mentioned please refer to comments in code blocks, as these explain specific selections.

```
1. // This method updates an organisms awareness, taking into account the terrain's visibility
2. private void updateAwareness(TerrainAttributes terrainAttributes){
3.     // Calls the method to check if the organism has encountered another organism
4.     checkOrganismProximity(stats.getCurrentAwareness());
5.     // Sets the variables that are read throughout
6.     double terrainVisibility = terrainAttributes.visibility; // The visibility attribute
for each terrain
7.     double organismsEncountered = stats.getOrganismsEncountered(); // The updated number of
organisms encountered
8.
9.     List<Double> awarenessHistory = stats.getAwarenessHistory(); // The list of previous
organism awareness'
10.    int awarenessHistoryArraySize; // Required as the size of the array is not always >= 4
11.
12.    // This switch expression sets awarenessHistoryArraySize and totalHistory variables.
13.    // Total history is a total of previous awareness', it is used to get the average
awareness (part of the algorithm)
14.    // awarenessHistoryArraySize is dependent on how old the organism is, therefore needs
to also be set depending on
15.    // the size of the array stored in stats.
16.    double totalHistory = switch (awarenessHistory.size()) {
17.        case 0 -> {
18.            awarenessHistoryArraySize = 1;
19.            yield 0.0;
20.        }
21.        case 1 -> {
22.            awarenessHistoryArraySize = 2;
23.            yield awarenessHistory.get(0);
24.        }
25.        case 2 -> {
26.            awarenessHistoryArraySize = 3;
27.            yield awarenessHistory.get(0) + awarenessHistory.get(1);
28.        }
29.        case 3 -> {
30.            awarenessHistoryArraySize = 4;
31.            yield awarenessHistory.get(0) + awarenessHistory.get(1) +
awarenessHistory.get(2);
32.        }
33.        default -> {
34.            awarenessHistoryArraySize = 5;
35.            yield awarenessHistory.get(0) + awarenessHistory.get(1) +
awarenessHistory.get(2) + awarenessHistory.get(3);
36.        }
}
```

```

37.         };
38.
39.         // Calculates the awareness, which is set to 2 decimal places (to ensure no long
40.         // decimals)
41.         double awareness = Double.parseDouble(decimalFormat.format(terrainVisibility +
organismsEncountered));
42.         // Calculates the organisms new currentAwareness. Set to 2 decimal places (to ensure no
long decimals)
43.         double currentAwareness = Double.parseDouble(decimalFormat.format((awareness +
totalHistory) / awarenessHistoryArraySize) + organismsEncountered));
44.         // Both use same calculations as logic in documentation (see 3.3.1.1)
45.
46.         stats.setCurrentAwareness(currentAwareness); // Updates the local stats variable
47.         awarenessHistory.add(0, awareness); // Adds the awareness to the locally held
48.         awarenessHistory = awarenessHistory.subList(0, Math.min(awarenessHistory.size(), 4));
// Restricts the size of awarenessHistory to 4
49.         stats.setAwarenessHistory(awarenessHistory); // Updates the awarenessHistory which is
held in JSON
50.     }

```

Java

The final method for this section is checkOrganismProximity, which as mentioned updates the number of organism encounters before awareness is updated. See below for this section of code, and as always refer to the comments for specific explanations.

```

1.     private void checkOrganismProximity(double doubleAwareness) {
2.         // Multiplies by 10 so that the bounds are 10px instead of 1
3.         int currentAwareness = (int) (doubleAwareness * 10); // Passing the awareness into an
integer, so it can be compared to coordinates.
4.         for (Organism organism : organisms) {
5.             if (organism.getOrganismId() != this.organismID) { // So it does not compare to its
own (would always be true)
6.                 int checkX = organism.getxCoordinate(), checkY = organism.getyCoordinate(); // The
coordinates to check
7.                 int xDifference = Math.abs(this.currentX - checkX); // Returning the difference
between mouse and organism coordinates
8.                 int yDifference = Math.abs(this.currentY - checkY); // Returns as an absolute
number, so it can be compared to only one bound
9.                 // If the coordinates are within the bound, then increase the number of
organisms encountered
10.                if (xDifference <= currentAwareness && yDifference <= currentAwareness) { // Bound is the
organisms awareness
11.                    stats.setOrganismsEncountered(stats.getOrganismsEncountered() + 0.1); // Increase the
number of organisms encountered
12.                }
13.            }
14.        }
15.    }

```

Java

3.3.1.3 Testing and fixing

Through adding output statements at various stages, the program correctly calculates the organisms' awareness', however organisms that have a lot of encounters enter a loop of consistently increasing awareness, even if they are not encountering any more organisms. This is due to there being no decrease in awareness. As seen in the image below, one organism which spawned next to another organism, in a water biome (which should have a hinderance to awareness), had an awareness of 21.8 (218 pixels) after 24 years. This was exponentially increasing, as with the higher the range the more organisms interacted with, and therefore a bigger increase each year.

The fix for this was simple, with the organisms encountered not having a limit and therefore causing a bigger increase each time. By adding a cap to the number of encounters, the awareness can now not go any higher than 2.2 (22-pixel radius).

Stats:

- Age: 24
- Size: 10
- Health: 100.0
- Speed: 1.0
- Awareness: 218.7px

```

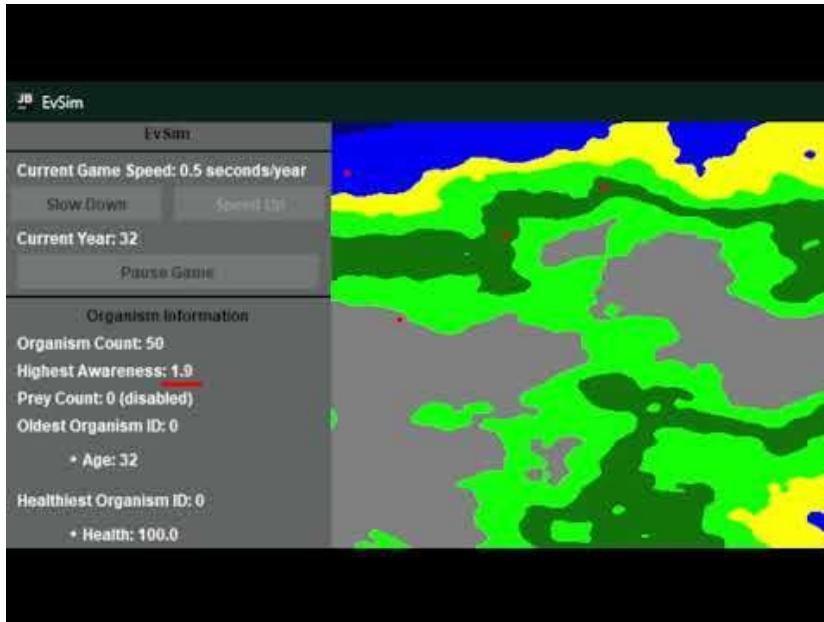
1.     stats.setOrganismsEncountered(Math.min(0.5, stats.getOrganismsEncountered()));
2.     // Hard-cap the amount of organisms encountered
3.     stats.setOrganismsEncountered(Math.max(0.0, stats.getOrganismsEncountered()));
4.     // Ensure it's not negative

```

Java

The method above is how organisms encountered has been hard capped, and also never a negative number (to avoid errors).

The video below shows the tracking of the organism with the highest awareness over 50 years.



This demonstrates that the awareness is correctly updating to the desired values, with the organism here encountering the maximum of 5 organisms, as well as being on a terrain which has a high visibility. The JSON below is linked with the organism in question.

```

1. {
2.     "organismId": 47,
3.     "stats": {
4.         "currentAwareness": 2.1,
5.         "organismsEncountered": 0.5,
6.         "awarenessHistory": [
7.             1.6,
8.             1.6,
9.             1.6,
10.            1.6
11.        ],
12.    },
13.    "currentTerrain": "Beach",
14.    "xCoordinate": 23,
15.    "yCoordinate": 435
16. },

```

JSON

As seen from this the organism is on a beach terrain, which has a visibility of 1.1. Counting that alongside the organisms encountered, the awareness is 1.6, as seen from awarenessHistory.

$$currentAwareness = \frac{1.6 + 1.6 * 4}{4 + 1} + 0.5 = 2.1$$

As seen with the above calculation, current awareness calculated by the program is correct, therefore proving this section to be finished. Once organism breeding is developed, this will be slightly changed in the fact that offspring will take attributes from the parent organisms.

3.3.2 Food

3.3.2.1 Logic

This section focuses on the food that is available to each organism in the different terrains. For example, a forest terrain will have a high proportion of food availability, with a small food factor. Food factor will be how much the food contributes to the organism's evolution. This is important as it reflects the change that it has on size and speed of an organism. For example, in a forest, food is widely available, with most food having a lower food factor, due to both its availability and density. On the other hand, the deep-water biome will have very little to no food at all, however if an organism was to find food, it would have a significant effect on the organism's evolution (food factor), giving it a bias towards size, speed and retention, all factors that would allow it to find food easier. Each terrain is shown below with the individual food types, along with the availability and food factor.

Grass:

- Grass seeds, Availability – 0.60, Food Factor – 0.30
- Leaves and larger plant types, Availability – 0.25, Food Factor - 0.40
- Tree Roots and plant remains found underground, Availability - 0.15, Food Factor - 0.5

Forest:

- Tree spawn such as acorns, Availability – 0.40, Food Factor – 0.25
- Berries, Availability – 0.50, Food Factor - 0.15
- Fungi, Availability - 0.30, Food Factor – 0.30

Water:

- Aquatic plants, Availability – 0.50, Food Factor – 0.20
- Algae, Availability – 0.40, Food Factor – 0.25
- Insects and Insect Larvae, Availability - 0.20, Food Factor - 0.50

Beach:

- Beach Grass Seeds, Availability - 0.40, Food Factor - 0.25
- Shellfish and Crustaceans, Availability – 0.30, Food Factor - 0.35
- Coastal Fruit/plants, Availability - 0.30, Food Factor - 0.40

Deep Ocean:

- Plankton, Availability - 0.30, Food Factor - 0.10
- Deep-sea Fish, Availability – 0.10, Food Factor - 0.50
- Large Deep-Sea Fish, Availability – 0.05, Food Factor - 0.80

Mountain:

- Grasses and Herbs, Availability – 0.40, Food Factor - 0.25
- Shrubs, Availability – 0.30, Food Factor - 0.35
- Coniferous Tree Spawn, Availability – 0.30, Food Factor - 0.40

Snow:

- Moss/Lichen, Availability – 0.40, Food Factor - 0.20
- Algae, Availability 0.30, Food Factor - 0.35
- Shrub Berries, Availability 0.30, Food Factor - 0.45

Each of these have been tailored to the terrain to meet ideal survivability rates of these biomes. For example, as planned with the deep-water terrain, food is scarce but rewarding, like in the case of Large Deep-sea fish. Whereas a forest has significantly higher rates of food, but with less of a reward, for example berries, which have a 50% availability, and a 15% food factor. These factors will be added to the relating Terrain class.

These food types will have classes, for example, plant spawn such as grass seeds and fungi can be grouped together, as can aquatic life, such as shellfish, plankton and insect larvae. See below for the full classification of food types:

Plant Spawn:

- Grass Seeds.
- Beach Grass Seeds.
- Grasses and Herbs
- Fungi

Plant Parts:

- Leaves and larger plant types.
- Tree Roots and plant remains found underground.

Fruits and Berries:

- Berries
- Coastal Fruit/plants
- Shrub Berries

Algae and Aquatic Plants:

- Aquatic plants
- Algae

Aquatic Life/Spawn:

- Insects and Insect Larvae
- Shellfish and Crustaceans
- Plankton

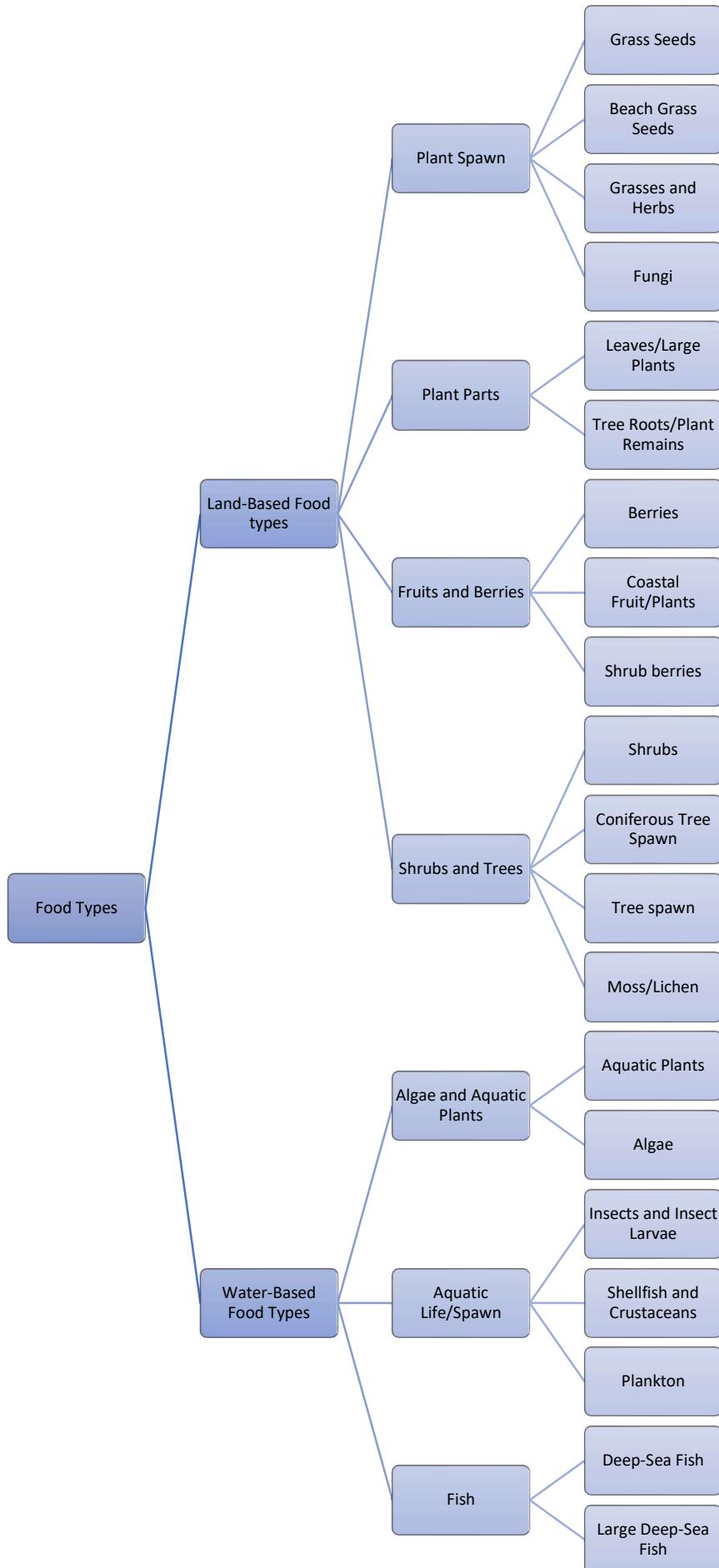
Fish:

- Deep-sea Fish
- Large Deep-Sea Fish

Shrubs and Trees:

- Shrubs
- Coniferous Tree Spawn
- Tree spawn.
- Moss/Lichen

These can also be filtered down into larger classes, for example Land-based food types, like Shrubs and Trees, Fruits and Berries, Plant Parts, Plant spawn, and Water-based food types, such as Fish, Aquatic Life/Spawn, Algae and Aquatic Plants. This means that food can be classified within the project code, and the organisms themselves are not restricted to one type of food. Food classes such as Plant Spawn can be found across multiple biomes, meaning an organism that moves out of its spawn biome will not die of starvation immediately. This also allows food to give a different food factor for an organism. For example, if an organism has spent 50 years eating plant-based food, but now finds itself in water, where it cannot eat plant-based food, then the food factor is so low for the organism that it will seek out land. This accurately matches what is seen in real life. See below for a hierarchy tree of food that is available within EvSim.



To outline the organism's choosing of food, once an organism spawns, for the first 5 years of an organism's life, they will eat at random from the biome that they are in, with the chance for each food being equal to the availability. For example, an organism in its first year in a mountain terrain will have a 40% chance to eat grasses and herbs, 30% chance to eat from shrubs, and 30% chance to eat coniferous tree spawn. This will continue over the span of the 5 years, whereby after it will then choose to seek out the class of food that it has consumed the most (preferred food type). This means that if the organism moves out of its spawn biome after the 5 years, it will seek out the class it has chosen to be its preferred food type, in this case Fruit and Berries or plant spawn. The organism can still eat from other food classes, but the food factor will be significantly reduced, encouraging the organism to find its preferred food source. **The further away from the preferred food, the less the food contributes to the organism.** For example, if the organism prefers to eat Tree Roots/Plant Remains, which has a food factor of 0.50, anything else within the plant parts class will contribute 80% of the original value, anything in other Land-Based Food types will contribute 60% of the original value, and anything in Water-Based Food Types will contribute 30% of the original value. Therefore, if the same organism eats Large Deep-Sea fish, the food factor for that will only contribute 0.24 (30% of 0.80).

All above accurately represents evolution as organisms actively seek their preferred food, but they are not being restricted to that food only. Organisms will benefit from eating their preferred food type more, as it will contribute to their size, speed as well as survivability ratio.

3.3.2.2 First implementation

To implement this into the project, I will make use of depth-first (DFS) graph traversal. I will replicate the tree found in 3.3.2.1, and then find the distance between 2 nodes. The distance between these nodes can be used to find the contribution of the food that is eaten. If the depth of the travel is within one branch, then the food contribution is 80% of the original, if it is within the same food type (land or water), it is 60%, and if it goes into the other food type then it is 30%. The algorithm to find distance between two nodes on a tree is shown below.

$$Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2 * Dist(root, lca)$$

'n1' and 'n2' are the two given keys. 'root' is root of given Binary Tree. 'lca' is lowest common ancestor of n1 and n2. Dist(n1, n2) is the distance between n1 and n2.

To find the lowest common ancestor, we will perform a DFS for both nodes, comparing the results, and removing all but the last common value, we will then have 2 lists that stem from the lowest common ancestor. See below for the resultant code, refer to comments for explanations of specific sections.

```

1.  public static List<String> dfs(TreeNodes root, String target) {
2.      List<String> visited = new ArrayList<>(); // Create a list to store visited nodes
3.      if (root != null) { // Check if the root exists before starting the DFS
4.          dfsHelper(root, visited, target); // Call dfsHelper which traverses the tree
5.      }
6.      return visited;
7.  }
8. // Traverses the tree from the root
9. private static void dfsHelper(TreeNodes node, List<String> visited, String target) {
10.     // If the object is not the one to be found and if it has not been found already
11.     if(!Objects.equals(node.val, target) && !found) {
12.         visited.add(node.val); // Add the current node to the visited list
13.         for (TreeNodes child : node.children) { // Explores the children of the current
node
14.             dfsHelper(child, visited, target); // If the child has children, explore those
15.         }
16.     } else {
17.         found = true; // It is the object to be found, ends the DFS
18.     }
19. }
20. // Removes all items from the list up until the lowest common ancestor

```

```

21.     public static void findLCA(List<String> list1, List<String> list2){
22.         // If the first value in both lists are the same
23.         if(Objects.equals(list1.get(0), list2.get(0)) && list1.size() != preferredFood.foodClassSize + 2 && list2.size() != eatenFood.foodClassSize + 2){
24.             list1.remove(0); // Remove the first values (as it is not the LCA)
25.             list2.remove(0);
26.             findLCA(list1, list2); // Recalls it with the new list
27.         } else {
28.             System.out.println(list1 + " + " + list2); // Output statement to test if LCA was
29.             found
30.         }

```

Java

To test to see if both the search and the method to find the LCA have worked, I will output the list which is returned by DFS, and the list that is left after LCA is executed. The preferred food will be set to Algae, and the food that is eaten set to Plankton.

The expected output for a DFS search for Algae is below:

Food Types -> Land Based Food -> Plant Spawn-> Grass Seeds -> Grasses and Herbs -> Beach Grass Seeds-> Fungi -> Plant Parts -> Leaves / Large Plants -> Tree Roots / Plant Remains -> Fruits and Berries -> Berries -> Coastal Fruit / Plants -> Shrub Berries -> Shrubs and Trees -> Shrubs -> Coniferous Tree Spawn -> Tree Spawn -> Moss / Lichen -> Water Based Food -> Algae and Aquatic Plants -> Aquatic Plants -> Algae

The expected output for a DFS search for Plankton is below:

Food Types -> Land Based Food -> Plant Spawn-> Grass Seeds -> Grasses and Herbs -> Beach Grass Seeds-> Fungi -> Plant Parts -> Leaves / Large Plants -> Tree Roots / Plant Remains -> Fruits and Berries -> Berries -> Coastal Fruit / Plants -> Shrub Berries -> Shrubs and Trees -> Shrubs -> Coniferous Tree Spawn -> Tree Spawn -> Moss / Lichen -> Water Based Food -> Algae and Aquatic Plants -> Aquatic Plants -> Algae -> Aquatic Life / Spawn -> Insects and Insect Larvae -> Shellfish and Crustaceans -> Plankton

The LCA is highlighted in blue within both lists. The program should return both of these lists, as well as both of these lists from the LCA. Shown below is the result from the console.

File - Main

```

1 "C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\lib\idea_rt.jar=53517:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 [Food Types, Land Based Food, Plant Spawn, Grass Seeds, Grasses and Herbs, Beach Grass Seeds, Fungi, Plant Parts, Leaves / Large Plants, Tree Roots / Plant Remains, Fruits and Berries, Berries, Coastal Fruit / Plants, Shrub Berries, Shrubs and Trees, Shrubs, Coniferous Tree Spawn, Tree Spawn, Moss / Lichen, Water Based Food, Algae and Aquatic Plants, Aquatic Plants, Algae, Aquatic Life / Spawn, Insects and Insect Larvae, Shellfish and Crustaceans, Plankton] + [Food Types, Land Based Food, Plant Spawn, Grass Seeds, Grasses and Herbs, Beach Grass Seeds, Fungi, Plant Parts, Leaves / Large Plants, Tree Roots / Plant Remains, Fruits and Berries, Berries, Coastal Fruit / Plants, Shrub Berries, Shrubs and Trees, Shrubs, Coniferous Tree Spawn, Tree Spawn, Moss / Lichen, Water Based Food, Algae and Aquatic Plants, Aquatic Plants, Algae]
3 [Water Based Food, Algae and Aquatic Plants, Aquatic Plants, Algae, Aquatic Life / Spawn, Insects and Insect Larvae, Shellfish and Crustaceans, Plankton] + [Water Based Food, Algae and Aquatic Plants, Aquatic Plants, Algae]

```

Line 2 shows the 2 lists, separated by a '+'. Comparing the expected result to what was returned, this shows that the DFS search is correct. Line 3 shows the lists from the LCA only, with both lists starting at the expected LCA (Water Based Foods), this validates these methods.

The method findLCA can be altered to return the index of the LCA only, sending this to the variable LCA. The equation displayed previously can now be filled:

$$Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2 * Dist(root, lca)$$

$$Dist(Algae, Plankton) = Dist(root, Algae) + Dist(root, Plankton) - 2 * Dist(root, LCA)$$

And therefore can be implemented into the program:

```

1.  private static boolean found = false;
2.  public static FoodTypes preferredFood = new Plankton(); // Preferred Food
3.  public static FoodTypes eatenFood = new Algae(); // Current Food
4.  public static List<String> result;
5.  public static List<String> secondResult;
6.  public updateFood() {
7.      // Brings in the tree
8.      TreeNodes treeNode = TreeNodes.root;
9.      // Perform DFS
10.     result = dfs(treeNode, preferredFood.foodName);
11.     secondResult = dfs(treeNode, eatenFood.foodName);
12.     // Adds the two food names to the end of the lists
13.     result.add(preferredFood.foodName);
14.     secondResult.add(eatenFood.foodName);
15.     // Finds the lowest common ancestor
16.     String LCAtoFind = findLCA(result, secondResult);
17.     int LCA = dfs(treeNode, LCAtoFind).size(); // Distance from root to LCA
18.     int dist1 = result.size(); // Distance from root to preferred food
19.     int dist2 = secondResult.size(); // Distance from root to eaten food
20.
21.     // Applies the formula to find the distance between nodes
22.     // Dist(n1,n2) = Dist(root,n1) + Dist(root,n2) - 2*Dist(root,lca)
23.     int distanceBetweenNodes = dist1 + dist2 - 2 * LCA;
24.
25.     // Outputting the result
26.     System.out.println("Distance between nodes " + preferredFood.foodName + " and " +
eatenFood.foodName + ": " + distanceBetweenNodes);
27. }
```

Java

It was during testing of this section where I noticed a fatal error with the depth first search. The DFS I was using was returning the full search, up until when the food was found, however the distances which were required within the algorithm require the best route. This would require each factor of the list to be compared, whether it was a parent class or not, adding complexity to the solution. With the DFS itself being complex and demanding on the system, I began to look for alternative solutions.

3.3.2.3 Second implementation

Considering the tree from 3.3.2.1, and the fact that it is a static tree which does not change throughout the running of the program, it would be better to alter the food classes to contain what their parent classes are. This therefore can be used to determine if two foods are under the same parent or not. This by definition is finding the LCA of two nodes, where the parents are known. The logic would go as follows:

```

if preferredFood.foodName == eatenFood.foodName then foodFactor = eatenFood.foodFactor
else if preferredFood.parentName == eatenFood.parentName then foodFactor = eatenFood.foodFactor * 0.8
else if preferredFood.waterBasedFood && eatenFood.waterBasedFood then foodFactor = eatenFood.foodFactor * 0.6
else foodFactor = eatenFood.foodFactor * 0.3

```

pseudocode

This is much simpler than performing a DFS for two lists, then finding the LCA of both, to factor all returned values into a formula. This will require significant expansion to food classes, with each one looking like the following.

```

1. package FoodTypes;
2. public class ShellfishCrustaceans extends FoodTypes {
3.     public ShellfishCrustaceans(){}

```

```

4.     foodName = "Shellfish and Crustaceans";
5.     foodAvailability = 0.3;
6.     foodFactor = 0.35;
7.     landBasedFood = false;
8.     waterBasedFood = true;
9.     terrainName = "Beach";
10.    parentName = "Aquatic Life / Spawn";
11. }
12. }
```

Java

One method which will be required, is finding the relevant class for both food types. This will follow a similar principle to the method for getting the correct terrain in 3.2.3. See below for this class.

```

1.  private FoodTypes getFoodType(String foodName) {
2.      FoodTypes foodType;
3.
4.      foodType = switch (foodName) {
5.          case "Grass Seeds" -> new GrassSeeds(); // Returns GrassSeeds food type
6.          case "Beach Grass Seeds" -> new BeachGrassSeeds(); // Returns BeachGrassSeeds
7.          case "Grasses and Herbs" -> new GrassesHerbs();
8.          case "Fungi" -> new Fungi();
9.          case "Leaves / Large Plants" -> new LeavesPlants();
10.         case "Tree Roots / Plant Remains" -> new RootsPlantRemains();
11.         case "Berries" -> new Berries();
12.         case "Coastal Fruit / Plants" -> new CoastalFruitPlants();
13.         case "Shrub Berries" -> new ShrubBerries();
14.         case "Shrubs" -> new Shrubs();
15.         case "Coniferous Tree Spawn" -> new ConiferousTreeSpawn();
16.         case "Tree Spawn" -> new TreeSpawn();
17.         case "Moss / Lichen" -> new MossLichen();
18.         case "Aquatic Plants" -> new AquaticPlants();
19.         case "Algae" -> new Algae();
20.         case "Insects and Insect Larvae" -> new InsectsLarvae();
21.         case "Shellfish and Crustaceans" -> new ShellfishCrustaceans();
22.         case "Plankton" -> new Plankton();
23.         case "Deep Sea Fish" -> new DeepSeaFish();
24.         case "Large Deep Sea Fish" -> new LargeDeepSeaFish();
25.         default -> new FoodTypes();
26.     };
27.     return foodType;
28. }
```

Java

This class is considerably lengthy and repetitive, however is much less demanding and therefore has a faster execution. The full updateFood class is now shown below.

```

1.  public void updateFood() {
2.      preferredFood = getFoodType("Grass Seeds");
3.      eatenFood = getFoodType("Moss / Lichen");
4.      double foodFactor;
5.
6.      // If the two foods are the same
7.      if (Objects.equals(preferredFood.foodName, eatenFood.foodName)) foodFactor =
eatenFood.foodFactor;
8.      else if (Objects.equals(preferredFood.parentName, eatenFood.parentName)) foodFactor =
eatenFood.foodFactor * 0.8; // If they are part of the same food class
9.      else if ((preferredFood.waterBasedFood && eatenFood.waterBasedFood) ||
(preferredFood.landBasedFood && eatenFood.landBasedFood)) foodFactor = eatenFood.foodFactor * 0.6;
// If they are the same food type
10.     else foodFactor = eatenFood.foodFactor * 0.3; // If they are different food types
11.
12.     System.out.println("Food factor between Grass Seeds and Moss / Lichen: " + foodFactor);
13.     System.out.println("Original value: " + eatenFood.foodFactor);
14. }
```

Java

As shown in the code, I will test this using the difference between 4 eatenFoods, the first one being Shellfish and Crustaceans, the second being Moss/Lichen, the third being Beach Grass Seeds and the final being Grass seeds. The test data for this is laid out in a table below, followed by the test results.

3.3.2.4 Testing food factor

The following formula is what is used within the calculation section of the testing table.

$$foodFactor = originalFactor * x$$

Test data for foodFactor				
preferredFood	x	eatenFood	Expected foodFactor	Calculation
Grass Seeds	0.30	Shellfish and Crustaceans	0.105	0.105 = 0.35 x 0.30
Grass Seeds	0.60	Moss / Lichen	0.12	0.06 = 0.2 x 0.60
Grass Seeds	0.80	Beach Grass Seeds	0.2	0.2 = 0.25 x 0.8
Grass Seeds	1.00	Grass Seeds	0.3	0.3 = 0.3 x 1

Refer to the output statements for each of the following screenshots, these detail which food factors were tested, as well as both the originalFactor and the returned value.

Grass Seeds and Shellfish and Crustaceans:

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50703:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Food factor between Grass Seeds and Shellfish: 0.105
3 Original value: 0.35
```

Grass Seeds and Moss / Lichen:

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50922:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Food factor between Grass Seeds and Moss / Lichen: 0.12
3 Original value: 0.2
```

Grass Seeds and Beach Grass Seeds:

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50968:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Food factor between Grass Seeds and Beach Grass Seeds: 0.2
3 Original value: 0.25
4
```

Grass Seeds and Grass Seeds:

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=50959:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Food factor between Grass Seeds and Grass Seeds: 0.3
3 Original value: 0.3
4
```

These tests show that the method here can provide the correct foodFactor when given the preferred food and eaten food as a string.

3.3.2.5 Further logic and implementation

Now that the food factor is calculated, the rest of the food attribute can be completed. This involves linking the food types with their appropriate terrains, as well as the organism's recognition and eating of food.

As with 3.3.1 awareness, updating the JSON to contain the information needed was the first step. Within the JSON, 4 new values will need to be stored per organism, preferredFood (FoodType), eatenFood (FoodType), recentFood (List<FoodType>) and foodFactor (double). recentFood will be a list of foodTypes which have been eaten recently. This will only be relevant for the first 5 years of an organism's life, so therefore will only be filled during that period. If the organisms age is greater than 5, then the recentFood can be set to an empty array. Otherwise, it will be used to store the recently eaten food. On the fifth year, as planned in 3.3.2.1 Logic, the organism will choose a preferred food from this array (the food that has been eaten the most), which will then be stored in the preferredFood key. The food that is eaten after the fifth year will now be stored in eatenFood (instead of recentFood), and the food factor (3.3.2.3) can now be found.

```
1. // Finding the food types for the current terrain
2.     ArrayList<String> availableFoods = terrainAttributes.foods;
3.
4.     if(stats.getAge() < 5){
5.         // If the organism is younger than 5, as by documentation the organism can eat
anything
6.         // What the organism eats within the first 5 years is added to a list of recent
foods
7.         // Once the organism passes year 5, they eat the food they ate the most within the
first 5 years
8.         String tempFood; // The food that will be chosen
9.         FoodTypes potentialFood1 = getFoodType(availableFoods.get(0)), potentialFood2 =
getFoodType(availableFoods.get(1)), potentialFood3 = getFoodType(availableFoods.get(2)); // The
foods that can be chosen
10.
11.        double totalAvailability = potentialFood1.foodAvailability +
potentialFood2.foodAvailability + potentialFood3.foodAvailability;
12.        double randomChoice = random.nextDouble(0, totalAvailability);
13.        if(randomChoice > 0 && randomChoice <= potentialFood1.foodAvailability) tempFood =
availableFoods.remove(0); // Returns the first item from the list
14.        else if (randomChoice > potentialFood1.foodAvailability && randomChoice <=
potentialFood2.foodAvailability + potentialFood1.foodAvailability) tempFood =
availableFoods.remove(1); // Returns the second item from the list
15.        else tempFood = availableFoods.remove(2); // Returns the last food in the list
16.
17.        List<FoodTypes> recentFoods = new ArrayList<>();
18.
19.        if (stats.getRecentFood() != null) recentFoods = stats.getRecentFood();
recentFoods.add(getFoodType(tempFood)); // Adds the most recent food
20.        stats.setRecentFood(recentFoods); // Updates the food eaten list
21.
```

```

22.           stats.setEatenFood(getFoodType(tempFood)); // The food which was selected is set to
as eaten
23.       } else if (stats.getAge() == 5) {
24.           // Once the organism passes year 5, they select preferred food based on the most
common food found during first 5 years
25.           List<FoodTypes> recentFoods = stats.getRecentFood();
26.           int count = 0;
27.           FoodTypes mostCommon = new FoodTypes();
28.           // Choosing the most common food that was eaten
29.           for (int counter1 = 0; counter1 < recentFoods.size(); counter1++) {
30.               int counter3 = 0;
31.               for (FoodTypes recentFood : recentFoods) if (recentFoods.get(counter1) ==
recentFood) counter3++;
32.               if (counter3 > count) {
33.                   count = counter3;
34.                   mostCommon = recentFoods.get(counter1);
35.               }
36.           }
37.           stats.setPreferredFood(mostCommon); // Set the preferred food to the food type of
most eaten food
38.           stats.setEatenFood(mostCommon); // Set the eaten food to the food type of most
eaten food
39.           stats.setRecentFood(new ArrayList<>()); // Clear the array to free space
40.       } else {
41.           // If the age is over 5, get the preferred food. Check if the preferred food is
part of the available foods list,
42.           // if it is set to the eaten food, if it isn't, use foodAvailability to set a new
eaten food from then environment
43.           // preferred and eaten food below
44.           if (availableFoods.contains(stats.getPreferredFood().foodName))
stats.setEatenFood(stats.getPreferredFood()); // If the preferred food is available, then eat it
45.           else {
46.               // If preferred food is not available, check each food available for the
closest match to the preferred food
47.               for (String food : availableFoods) {
48.                   FoodTypes currentFood = getFoodType(food);
49.                   if (currentFood.parentName.equals(stats.getPreferredFood().parentName)) {
50.                       stats.setEatenFood(currentFood); // Within the same food category,
therefore closest
51.                       break;
52.                   } else if (currentFood.landBasedFood ==
stats.getPreferredFood().landBasedFood) {
53.                       stats.setEatenFood(currentFood); // Within same food type, continue
checking as may not be same class
54.                       if (food.equals(availableFoods.get(availableFoods.size() - 1))) break;
// If there are no more available foods, then this is the closest
55.                   } else stats.setEatenFood(getFoodType(food)); // If all available food
types are not within the same food type or category
56.               }
57.           }
58.       }

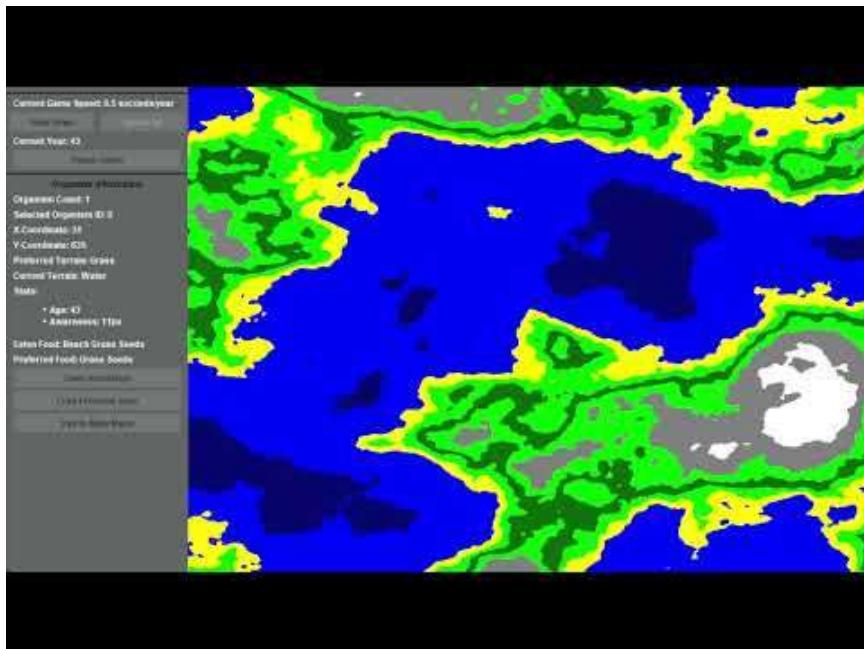
```

Java

Please refer to comments within this code block, as they explain what each section does. As an organism does not have a preferred food during the first 5 years, the implementation from 3.3.2.3 was adapted to check if the food type was not null. If the preferred food type does not exist, then the food factor does not change, otherwise, the previous calculation takes place. The food factor attribute will become important for other sections of evolution, such as organism size and speed. The food selection process however is now complete, and therefore only testing is required to finish 3.3.2.

3.3.2.6 Final testing

This section will be relatively small, as the best way to test an organism's selection of food, is to record one organism's life span, and compare with expected values. See below for the video and data documenting this.



Within this video, the organism at the bottom left moves between Grass, Beach, Water and Deep Water. The organism's preferred food after the first 5 years was set to Grass Seeds, however with most of the years after, the organism was not within the grass terrain (movement is random) and therefore the organism has to select the closest food, which is what the basis for this test is.

As seen within the video, when the organism is on the beach terrain (displayed in yellow), then it eats Beach Grass Seeds (which is within the same food class – Plant Spawn). If the organism is in the water, then it eats Insects and Insect Larvae, which is not a land-based food type, therefore proving this test to be a success. When the organism was in a different biome with another of its food class available, it chose from there, whereas if the organism was in a biome with none of the same food type available, it chose from availability the food that was available.

3.3.3 Size

The organism's size is affected quite heavily by the organisms food factor, as well as the food they ate, and the terrain that they are in. The size of an organism should also change very slowly, over generations of organisms. Due to the fact that organism breeding isn't available yet, I will set up the size so that an organism grows slowly during its one life cycle. The size of an organism will be displayed to the user in a measurement of m^2 , with the default value of $0.5m^2$.

3.3.3.1 Logic

Ideally, the size of one organism, which can live infinitely (breeding not available), should change every ~75 years, by a very small amount. An organism which has lived for 75 years, which eats Large Deep-Sea fish (a high food factor), should begin to grow, but only around $0.1m^2$. This size is small enough for the organism to not grow extensively, but also to demonstrate that an organism is growing. For this to happen, I will add a new attribute to the stats array within JSON, one called sizeFactor. This double attribute will take the organisms food factor over the years, and if it passes 25.0, the organism will grow by $0.05m^2$. If the organism does not pass 25.0 within the 75 years, then the organism will decrease in size instead. This means that organisms that eat Berries (food factor 0.15) for most of its life will not have the food backing to grow any larger, therefore growing smaller. This means that any organisms eating food with a food factor of lower than 0.3 will begin to shrink.

3.3.3.2 Implementation

With sizeFactor added into the JSON, it needs to be updated every year. Within a method called updateFood, if the organism's age is not a factor of 75, then only the sizeFactor has to change during this iteration. The size factor will add on the eatenFood's food factor (see 3.3.3.1) to its current value, and then end the method.

If the organism's age is a factor of 75, then the sizeFactor will be checked against 25.0, if it is larger, then the organism will increase its size by 0.05, and if not, decrease by 0.05. This will repeat across the organism's life. Once organisms can reproduce, organisms will become more resistant to size change, finalising around a size which is appropriate to its preferred food.

The method for size change is shown below.

```

1.      // Update the organisms size
2.      // If an organism is 75 (or a factor of), then they change size - grow or shrink depending
on food factor
3.      // Else, the food factor is added onto a counter variable.
4.      public void updateSize(){
5.          if(stats.getAge() % 75 == 0 && stats.getAge() != 0){ // If the age is a factor of 75
and not 0
6.              if(stats.getSizeFactor() >= 25){ // If the total food factor is high
7.                  stats.setSize(stats.getSize() + 0.05); // Increase the size
8.                  stats.setSizeFactor(0.0); // Reset the total food factor
9.              } else { // Lower than the threshold for growth
10.                  stats.setSize(stats.getSize() - 0.05); // Decrease the size
11.                  stats.setSizeFactor(0.0); // Reset the total food factor
12.              }
13.          } else {
14.              stats.setSizeFactor(stats.getSizeFactor() + stats.getEatenFood().foodFactor); // Else, update the total food factor
15.          }
16.      }

```

Java

```

File - Main
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=49422:C:\Program Files\JetBrains\
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\
IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-
annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Increase
3 Decrease
4 Increase
5 Decrease
6 Decrease
7 Decrease
8 Decrease
9 Increase
10 Increase
11 Decrease
12 Increase
13 Decrease
14 Decrease
15 Decrease
16 Increase
17 Increase
18 Decrease
19 Increase
20 Increase
21 Decrease
22 Increase
23 Decrease
24 Increase
25 Decrease
26 Increase
27 Decrease
28 Decrease
29 Increase
30 Decrease
31 Decrease
32 Increase
33 Increase
34 Increase
35 Increase
36 Increase
37 Decrease
38 Decrease
39 Decrease
40 Decrease
41 Decrease
42 Increase
43 Decrease
44 Increase
45 Decrease
46 Increase
47 Increase
48 Decrease
49 Decrease
50 Decrease
51 Increase

```

3.3.3.3 Testing Size

Ideally for size to be correct, after the first 75 years, organisms should be split equally, those that decrease, and those that increase in size. In a simulation of 50 organisms, on the 75th year, 23 organisms had an increase in size, and 27 had a decrease in size. See left for the console log representing this.

Since in this simulation the proportion of organisms which increased in size compared to those that decreased in size was ~50%, it is accurate to say that these organisms have correctly changed size as a population, with more having less access to food, but there still being an equal proportion.

To test the individual organisms, I will prevent the recentFood array from being updated, instead, updating it with the 75 food sources throughout the years, before comparing it with the

expected result. Due to the resultant JSON being quite large, I have omitted a large portion of the array, marked by [...], however the reason for this will be explained below.

The organism which was tested had a preferred food type of Insects and Insect Larvae, and only stayed within the water biome for most of its life (excluding first 5 years). Due to this, I have removed the middle section of the recent foods array, as it was repetitive and did not contribute to the testing. Over the 75 years, the organism would have amassed a total food factor of 36.35, therefore placing it above the threshold, and growing the organism. As shown by the data below, the organism after 75 years had a size of 0.55, meaning it did grow as expected.

```
1. {
2.     "organismId": 0,
3.     "stats": {
4.         "currentAwareness": 0.6,
5.         "organismsEncountered": 0.0,
6.         "awarenessHistory": [
7.             0.6,
8.             0.6,
9.             0.6,
10.            0.6
11.        ],
12.        "age": 75,
13.        "preferredFood": {
14.            "foodName": "Insects and Insect Larvae",
15.            "foodAvailability": 0.2,
16.            "foodFactor": 0.5,
17.            "waterBasedFood": true,
18.            "landBasedFood": false,
19.            "parentName": "Aquatic life / Spawn",
20.            "terrainName": "Water"
21.        },
22.        "eatenFood": {
23.            "foodName": "Insects and Insect Larvae",
24.            "foodAvailability": 0.2,
25.            "foodFactor": 0.5,
26.            "waterBasedFood": true,
27.            "landBasedFood": false,
28.            "parentName": "Aquatic life / Spawn",
29.            "terrainName": "Water"
30.        },
31.        "recentFood": [
32.            {"foodName": "Insects and Insect Larvae"},  

33.            {"foodName": "Insects and Insect Larvae"},  

34.            {"foodName": "Aquatic Plants"},  

35.            {"foodName": "Algae"},  

36.            {"foodName": "Insects and Insect Larvae"},  

37.            {"foodName": "Insects and Insect Larvae"},  

38.            ["[...]",  

39.            {"foodName": "Insects and Insect Larvae"},  

40.            {"foodName": "Insects and Insect Larvae"}  

41.        ],
42.        "foodFactor": 0.5,
43.        "sizeFactor": 36.35,
44.        "size": 0.55
45.    },
46.    "preferredTerrain": "Water",
47.    "currentTerrain": "Water",
48.    "xCoordinate": 1487,
49.    "yCoordinate": 757,
50.    "isDead": false
51. }
```

JSON

This section is now complete, with the organisms growing slowly, and growing depending on their food. This section will be returned to once organism breeding is in place, as organisms will take attributes from their parents, which will offset the organism's starting size.

3.3.4 Speed

The speed of an organism will directly relate to the organism's ability to move around the map. Like size, speed will rely heavily on the food factor of the organism, however, will evolve faster. An organism at the start of its life will be able to move a maximum of 5px each year, depending on the organism's food consumption, their speed will increase and therefore so will their allowed movement distance.

3.3.4.1 Logic

After testing with some values, the formula to calculate an organism's speed is below.

$$\text{CurrentSpeed} = \text{oldSpeed} * (1.01 + \text{foodFactor} * \text{age} / (1 + \text{age}^3))$$

Within this formula, the speed increases each year depending on the foodFactor, and its age. The age is important, as the older an organism gets, the slower it can increase its speed. Test data for this formula, using an example organism life span, shows the organism to increase its movement from 5 to 10px over a span of 70 years. This seems small, however as organisms breed the increase will become larger, meaning a 4th generation organism may increase to a speed of 20 or 25px. Therefore, allowing evolution to be implemented. The test data used to generate this function is shown below.

Age:	Food Factor:	Current Speed	Difference	Age:	Food Factor:	Current Speed	Difference
0	0.3	5.00	0.00	36	0.35	10.07	0.10
1	0.35	5.93	0.93	37	0.15	10.17	0.10
2	0.35	6.45	0.52	38	0.35	10.28	0.10
3	0.15	6.61	0.17	39	0.35	10.38	0.11
4	0.3	6.80	0.19	40	0.2	10.49	0.11
5	0.35	6.96	0.16	41	0.35	10.59	0.11
6	0.35	7.10	0.14	42	0.35	10.70	0.11
7	0.8	7.29	0.19	43	0.35	10.81	0.11
8	0.35	7.40	0.11	44	0.35	10.92	0.11
9	0.35	7.51	0.11	45	0.35	11.03	0.11
10	0.35	7.61	0.10	46	0.35	11.14	0.11
11	0.15	7.69	0.09	47	0.3	11.26	0.11
12	0.35	7.79	0.10	48	0.35	11.37	0.11
13	0.35	7.88	0.09	49	0.35	11.49	0.12
14	0.35	7.98	0.09	50	0.3	11.60	0.12
15	0.5	8.07	0.10	51	0.3	11.72	0.12
16	0.35	8.16	0.09	52	0.35	11.84	0.12
17	0.35	8.26	0.09	53	0.35	11.96	0.12
18	0.35	8.35	0.09	54	0.25	12.08	0.12
19	0.25	8.44	0.09	55	0.15	12.20	0.12
20	0.35	8.53	0.09	56	0.35	12.32	0.12
21	0.35	8.62	0.09	57	0.2	12.45	0.12
22	0.3	8.71	0.09	58	0.35	12.57	0.13
23	0.3	8.80	0.09	59	0.35	12.70	0.13
24	0.25	8.90	0.09	60	0.35	12.83	0.13
25	0.35	8.99	0.09	61	0.35	12.96	0.13
26	0.35	9.08	0.09	62	0.2	13.09	0.13
27	0.35	9.18	0.10	63	0.3	13.22	0.13
28	0.35	9.28	0.10	64	0.35	13.35	0.13

29	0.35	9.37	0.10	65	0.25	13.49	0.13
30	0.35	9.47	0.10	66	0.35	13.62	0.14
31	0.5	9.57	0.10	67	0.35	13.76	0.14
32	0.15	9.67	0.10	68	0.35	13.90	0.14
33	0.35	9.77	0.10	69	0.35	14.04	0.14
34	0.35	9.87	0.10	70	0.35	14.18	0.14
35	0.35	9.97	0.10				

As shown above, the organism would end its 70 years on 14.18px, with an increase of 0.14 from the previous year. This organism had a relatively low food factor, if it were to eat something such as Tree Roots and Plant Remains (food factor of 0.5), then the resultant speed would be 16.69. This demonstrates that the organisms that eat the higher valued food develop faster.

3.3.4.2 Implementation

As always, the first part of implementing this attribute, is adding the required factors to JSON. For this attribute however, speed (double) is the only factor to be added. With speed now being available in JSON, the method can be created. This method is one of the smallest for organism attributes due to it only being a formula, however, it will be expanded during the introduction of breeding. See below for the resultant method and refer to comments for explanation.

```

1.    // Updates the organisms speed
2.    // Takes the old speed, age and foodFactor into a formula
3.    public void updateSpeed(){
4.        // newSpeed = oldSpeed * (1.01 + foodFactor * age / (1 + age ^ 3)
5.        double oldSpeed = stats.getSpeed();
6.        double foodFactor = stats.getFoodFactor();
7.        int age = stats.getAge();
8.        // Formatted to 2dp to make more readable and take less space in JSON
9.        double newSpeed = Double.parseDouble(decimalFormat.format(oldSpeed * (1.01 + foodFactor
* age / (1 + Math.pow(age, 3)))));
10.       System.out.println("Old Speed = " + stats.getSpeed() + " New Speed = " + newSpeed + "
Age: " + age); // To test
11.       stats.setSpeed(newSpeed); // Updating the stats
12.   }

```

Java

3.3.4.3 Testing speed

Testing here will be comparing the data from 3.3.4.1 with the output from 3.3.4.2. If an organism ends with a value around 14.18 after 70 years, then the speed can be considered complete. I will also compare the food type, to check that the higher food factors are not being undervalued.

File - Main

```

62 Old Speed = 11.46 New Speed = 11.58 Age: 60
63 Old Speed = 11.58 New Speed = 11.7 Age: 61
64 Old Speed = 11.7 New Speed = 11.82 Age: 62
65 Old Speed = 11.82 New Speed = 11.94 Age: 63
66 Old Speed = 11.94 New Speed = 12.06 Age: 64
67 Old Speed = 12.06 New Speed = 12.18 Age: 65
68 Old Speed = 12.18 New Speed = 12.3 Age: 66
69 Old Speed = 12.3 New Speed = 12.42 Age: 67
70 Old Speed = 12.42 New Speed = 12.54 Age: 68
71 Old Speed = 12.54 New Speed = 12.67 Age: 69
72 Old Speed = 12.67 New Speed = 12.8 Age: 70

```

This organism spent its 70 years in the Water biome, feeding off Aquatic Plants (food factor 0.2).

Considering the test data from 3.3.4.1, where the food factor averaged 0.35, and the data here, where the food factor averaged 0.2, speed can be considered correctly calculated here, as the speed increased at a reduced rate over the same time with a lower food factor. The expected result was given, with no issues in between.

3.3.5 Temperature

Considering the fact that the temperature which was included in the 3.2.3 Organism Evolution was suitable for the temperature, I will use the same logic from there, expanding it slightly to include a temperature factor which will count towards an organism's health and survivability calculations.

3.3.5.1 Logic

The temperatures for each terrain are listed below:

Grass – 16 – 22°

Beach – 13 – 23°

Forest – 20 – 25°

Mountain – 4 – 10°

Snow – -4 – 4°

Water – 6 – 13°

Deep Water – 0 – 4°

When an organism updates, it will need to fetch the temperature of that terrain. The structure here will follow along with food, where the first 5 years the organism can withstand any temperature, however after that, its preferred temperature will become an average of those. Depending on how close the current temperature is to the preferred temperature, is the temperature factor. The base level for the temperature factor is 1. The further away the current temperature is to the preferred temperature, the lower the temperature factor. For each degree of temperature away from the preferred, the temperature factor decreases by 0.04. This means that on average, the largest decrease in temperature an organism should see, is down to -0.16. This would only be if an organism from a forest, goes to deep water, within which it would not be able to survive in. The temperature factor represents that fact, alongside other factors such as food availability.

3.3.5.2 Implementation

To implement this logic into Java was relatively straightforward, due to many of the features of this section being replicated within food. For example, the organism being under 5 storing its recent temperatures, before selecting its preferred temperature. Due to the formula also being straight forward ($1 - \text{difference} * 0.04$), calculating the temperature factor was a small task. The code for the implementation is shown below, refer to comments for explanation of specific sections.

```
1. // This method updates the organisms preferred temperature as well as the temperature factor
2. // Takes in the current temperature of the terrain, as well as the preferred temp (if applicable)
3. public void updateTemperature(TerrainAttributes terrainAttributes){
4.     int newTemp = terrainAttributes.temperature; // The new temperature for the organism
5.
6.     if(stats.getAge() < 5){ // If the organism is younger than 5, it can handle any
temperature
7.         List<Integer> recentTemp = stats.getRecentTemp(); // Update the list of recent
temperatures
8.         recentTemp.add(newTemp); // Add in the new temperature
9.         stats.setRecentTemp(recentTemp); // Update the array in stats
10.    } else if (stats.getAge() == 5){ // If the organism is 5 years old, select a preferred
temperature
11.        List<Integer> recentTemp = stats.getRecentTemp(); // Gets the list of previous
temperatures
12.        int mostCommon;
13.        // Find the average temperature from the 5 previous years
14.        mostCommon = recentTemp.get(0) + recentTemp.get(1) + recentTemp.get(2) +
recentTemp.get(3) + recentTemp.get(4);
```

```

15.         mostCommon = mostCommon / 5;
16.         stats.setPreferredTemp(mostCommon); // Set that as the preferred temperature
17.         stats.setRecentTemp(new ArrayList<>()); // Clear the recent temperature array to
    save space
18.     } else { // Find the organisms temperature factor - based on the organisms preferred
        and current temperature
19.         int preferredTemp = stats.getPreferredTemp();
20.         int difference = Math.abs(newTemp - preferredTemp); // Finding the difference (as a
positive (absolute value)) between the preferred and the current
21.         stats.setTempFactor(1 - difference * 0.04); // Setting the new temperature factor
which will be used later.
22.     }
23. }
```

Java

3.3.5.3 Testing Temperature

Within this, I will test one organism. The organism's choice of temperature on the fifth year will be checked, as well as the organism's temperature factor in the years after. To test these attributes, relevant output statements were added to the relevant sections. See below for the console result after this test.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=53325:C:\Program Files\JetBrains\
[IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\
Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\
lib\jackson-core-2.13.3.jar Main
2 Recent Temperatures: [10, 8, 10, 6, 10]
3 Preferred Temp: 8
4 Temperature factor: 0.92
```

From this, the recent temperatures are being recorded as expected, with the average temperature also being taken ($10 + 8 + 10 + 6 + 10 = 44 / 5 = 8.8 = 8$).

For the temperature factor, with the newTemp at that stage being 6, and the preferred temperature being 8, the temperature factor is also correct – $(1 - \text{difference} / 2) * 0.04 = 0.92$.

3.3.6 Preferred Habitat

This section is very simple, as an organism's preferred habitat once selected is highly unlikely to change. However, the organism will also have a second and third most preferred habitats, which are more likely to change as an organism enters a new terrain.

3.3.6.1 Logic

This will only require a comparison of attributes such as the temperature and food. The preferred temperature will be compared to the temperatures of each terrain, if the preferred temperature is within the bounds of a terrain, it will get a point towards it. Similarly, the food will be compared with that of what is available in each terrain, if it contains it, it will get a point. To avoid having two terrains with an equal amount of points the organism can only select its preferred terrain from ones it has visited. For most organisms, this will be their spawn biome, as they are unlikely to move across terrains during the first 5 years. If still there are equal points, the organism will just select a random one, as both are equal to the organism.

3.3.6.2 Implementation

The first section of implementation covers the organism's choice of one habitat. Refer to the comments for an explanation of this.

```

1. // Compare the attributes of the current terrain to the organisms preferred attributes
2. // If the attributes are favourable, set it to the preferred
```

```

3.          // If the organism, already has a preferred habitat then compare the new values to the
old values,
4.          // if more favourable, then replace the preferred habitat, if not, ignore.
5.          if(stats.getPreferredTerrain() != terrainAttributes.terrainId) {
6.              if (stats.getAge() >= 5) { // If the organism has chosen its preferred attributes
7.                  int currentTemp = terrainAttributes.temperature;
8.                  List<String> availableFoods = terrainAttributes.foods;
9.                  int preferredTemp = stats.getPreferredTemp();
10.                 String preferredFood = stats.getPreferredFood().foodName;
11.                 int currentCount = stats.getCurrentCount();
12.
13.                 int newCount = 0;
14.                 if (availableFoods.contains(preferredFood)) {
15.                     newCount = newCount + 2; // If the food is contained in that terrain get an
extra point
16.                 } else newCount = newCount + 1;
17.
18.                 int tempDiff = Math.abs(preferredTemp - currentTemp);
19.                 if (tempDiff < 5) newCount = newCount + 3; // If the preferred and current
temperatures are within a range of 5 from each other get 3 points
20.                 else if (tempDiff < 10) newCount = newCount + 2; // Within 10 then 2 points
21.                 else newCount = newCount + 1;
22.
23.                 if (currentCount < newCount) { // If the attributes are more preferable
24.                     stats.setCurrentCount(currentCount);
25.                     stats.setPreferredTerrain(terrainAttributes.terrainId);
26.                 }
27.             } else { // If the organism is under 5 then default the preferred terrain to its
current terrain
28.                 stats.setPreferredTerrain(terrainAttributes.terrainId);
29.             }
30.         }

```

Java

The next section of this would be to check the organisms second and third most preferred terrains. For this I have created two more attributes in the JSON which hold the second and third most preferred terrain. These could be stored in an Array, however considering there are only 3 terrains that can be set, it would be much simpler programming wise to just store as 3 strings (TerrainAttributes class cannot be stored in JSON). The string (terrain) which is stored will be used to fetch the actual terrain attributes through the getTerrainAttributes method designed in 3.2.3.

```

1.      // Compare the attributes of the current terrain to the organisms preferred attributes.
2.      // If the attributes are more favourable than its other preferred terrains, add it as
preferred to the appropriate position
3.      public void updateHabitat(TerrainAttributes terrainAttributes) {
4.          String preferredTerrain = stats.getPreferredTerrain();
5.          String secondPreferredTerrain = stats.getSecondPreferredTerrain();
6.          String thirdPreferredTerrain = stats.getThirdPreferredTerrain();
7.          if (stats.getAge() >= 5 && !Objects.equals(currentTerrain, preferredTerrain) &&
!Objects.equals(currentTerrain, secondPreferredTerrain) && !Objects.equals(currentTerrain,
thirdPreferredTerrain)) { // If the organism has chosen its preferred attributes
8.              int newCount = newCount(terrainAttributes); // The current terrain's score
9.
10.             int currentCount = newCount(getTerrainAttributes(preferredTerrain)); // The
preferred terrain's score
11.             int secondCurrentCount = newCount(getTerrainAttributes(secondPreferredTerrain)); // The
score of the second and third terrains
12.             int thirdCurrentCount = newCount(getTerrainAttributes(thirdPreferredTerrain));
13.
14.             // Checks the current score compared to other preferred terrains
15.             if (currentCount < newCount) {
16.                 stats.setPreferredTerrain(terrainAttributes.terrainId);
17.                 stats.setSecondPreferredTerrain(preferredTerrain); // Moving the other items
down the list
18.                 stats.setThirdPreferredTerrain(secondPreferredTerrain);
19.             }
20.             else if (secondCurrentCount < newCount && newCount != currentCount) {
21.                 stats.setSecondPreferredTerrain(terrainAttributes.terrainId);

```

```

22.         stats.setThirdPreferredTerrain(secondPreferredTerrain); // Moving the other
items down the list
23.     }
24.     else if (thirdCurrentCount < newCount && newCount != secondCurrentCount)
stats.setThirdPreferredTerrain(terrainAttributes.terrainId);
25. } else stats.setPreferredTerrain(preferredTerrain); // Default to the current terrain
26. }
27.
28. // Adds up the score of the terrain sent in, is used to check against preferred terrains
29. private int newCount(TerrainAttributes terrainAttributes){
30.     if(!Objects.equals(terrainAttributes.terrainId, "Default")) { // If it is not set to
the TerrainAttributes (null attributes)
31.         int currentTemp = terrainAttributes.temperature;
32.         List<String> availableFoods = terrainAttributes.foods;
33.         int preferredTemp = stats.getPreferredTemp();
34.         String preferredFood = stats.getPreferredFood().foodName;
35.
36.         int newCount = 0;
37.         if (availableFoods.contains(preferredFood)) {
38.             newCount = newCount + 2; // If the food is contained in that terrain get an
extra point
39.         } else newCount = newCount + 1;
40.
41.         int tempDiff = Math.abs(preferredTemp - currentTemp);
42.
43.         if (tempDiff < 5) newCount = newCount + 3; // If the preferred and current
temperatures are within a range of 5 from each other get 3 points
44.         else if (tempDiff < 10) newCount = newCount + 2; // Within 10 then 2 points
45.         else newCount = newCount + 1;
46.         return newCount;
47.     }
48. }
49. }

```

Java

As always refer to the comments within the code for explanation. The counting of a terrains score was moved into a function (newCount) so that it could be reused efficiently. The function returns the score of the terrain if the attribute sent in is not default.

3.3.6.3 Testing

Due to the organism being unlikely to change its preferred terrain throughout its lifespan, it is not possible to test this at the current stage (will be tested once organisms can move and breed efficiently). However, the second and third preferred terrain can be tested, through checking of the current terrain's attributes and the organism's preferred attributes. See below.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=65271:C:\Program Files\JetBrains\
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\\
Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Most preferred: Forest Second Most Preferred Terrain: Default Third most preferred
terrain: Default Current terrain: Forest
3 Most preferred: Forest Second Most Preferred Terrain: Grass Third most preferred terrain
: Default Current terrain: Grass
4 Most preferred: Forest Second Most Preferred Terrain: Grass Third most preferred terrain
: Beach Current terrain: Beach

```

During a span of 123 years, this organism visited the forest (spawn), grass, and beach. Due to the organism spawning in a forest, the preferred terrain should be the forest. As seen from the console data, this is correct throughout. The grass terrain was visited next, due to its attributes being largely like the forest (forest inherits grass), this should be the second preferred terrain, and as seen in the console, this is correct. The final terrain which was visited here was the beach, and due to many of its variables

being different (one food type is a water-based food, temperature is much hotter), this should be the third most preferred terrain. In the console data, this is correct.

To further this test, I will continue the running until the organism visits a fourth terrain, this terrain should not be any of the preferred terrains, as the terrains which are closest to forest have already been selected.

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=65271:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Most preferred: Forest Second Most Preferred Terrain: Default Third most preferred terrain: Default Current terrain: Forest
3 Most preferred: Forest Second Most Preferred Terrain: Grass Third most preferred terrain : Default Current terrain: Grass
4 Most preferred: Forest Second Most Preferred Terrain: Grass Third most preferred terrain : Beach Current terrain: Beach
5 Most preferred: Forest Second Most Preferred Terrain: Grass Third most preferred terrain : Beach Current terrain: Water
```

As seen from line 5, the organism next visited the water terrain, and as expected, it was not set to any of the preferred terrains, marking this as a complete attribute.

3.3.7 Health

The organism's health will keep track of whether the organism is alive or not. The organism's health will decrease by default over a range of 70 years, meaning that if an organism does not have any negative affecting factors (such as a lack of food), they should die by default on the 70th year.

3.3.7.1 Logic

If the organism starts on 100 health, the health should by default decrease to 0 by the year 70, this means the value decreases by 1.4 each year. This factor can be influenced by world events, such as being attacked (predator/prey), or if the organism does not eat its preferred food / a lower valued food. If an organism does not eat its preferred food, then the value will decrease by an additional amount that year (calculated within food factor, see 3.3.2). If the current temperature is higher or lower from the preferred temperature, then the value decrease will also change by an additional amount (calculated within temperature factor, see 3.3.5). Therefore, if an organism which prefers a grass terrain is in deep water, its health will decrease significantly faster, as it is in an unfamiliar terrain in which it cannot survive.

3.3.7.2 Implementation and Testing

As mentioned in 3.3.7.1 Logic, the food factors and temperature factor will be used to affect health. For example, the food factor and temperature factor are measured as a decimal, these can be used to affect health. The further away an organism's temperature factor is from 1 (default) (see 3.3.5), is the additional amount taken away each year from the health. Likewise, the further away the food factor is from 1 (see 3.3.2), the more the health decreases. This means organisms which prefer to eat food with a lower food factor (like berries), will have a shorter life span due to the food not being as rewarding for the organism.

3.3.7.2.1 The organism's life span

This section will ensure the organism dies around the 70th year by default. The code for which is simple, each year the health will decrease by 1.4, once it is 0, the JSON attribute 'dead' will be set to true, and the organism will be painted black on the screen instead of red.

```

1.    // Updates the health of an organism
2.    private void updateHealth() {
3.        double currentHealth = stats.getHealth();
4.
5.        currentHealth = currentHealth - 1.4;
6.        if(currentHealth <= 0) {
7.            stats.setDead(true);
8.            System.out.println("Health: " + currentHealth + " Age: " + stats.getAge());
9.        } else if (currentHealth > 100) stats.setHealth(100.0);
10.       stats.setHealth(currentHealth);
11.   }

```

Java

In the code block above, the health decreases each time this method is called (each year), if the health becomes lower than 0, the organism is set to dead. If the health is above 100, it is set to 100. The latter part of that is extremely unlikely to be used, however it is there to enforce the bounds.

The output of this is shown below.

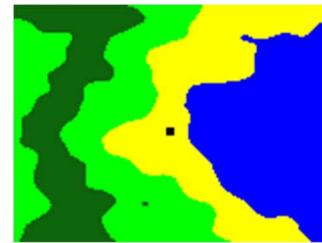
File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=65406:C:\Program Files\JetBrains\
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\
IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\
lib\jackson-core-2.13.3.jar Main
2 Health: -0.800000000000966 Age: 71

```

Oldest Organism ID: 0
• Age: 71
Healthiest Organism ID: 0
• Health: 0.599999999999033



As seen from both the console log (top) and the side menu in game (bottom), the organism reaches a health of 0 on the year 71 (note the organisms age increases after all the stats update, hence the discrepancy between the age and the health in both*). *The side menu updates after each update, the output statement within the console updates *before* the organisms age is updated, therefore on the side menu when the health is -0.8, the age would be 72. As seen on the right as well, the organism has turned black, indicating its death on the screen. Due to the long decimals, before the health is saved, I will implement a decimal factor (factoring it down to 2dp) which will ensure the removal of long decimals which are created because of Java's floating-point maths.

From the tests here, the organism successfully dies on the 72nd year of its life, meaning this section is complete.

3.3.7.2.2 Factors affecting health

This section covers how the temperature factor, food factor, and terrain choice affect health. As mentioned in 3.3.7.1 Logic, the health will decrease depending on the food and temperature factor. The decrease for each will be equal to 1 less the factor. For temperature, if the organism's current temperature is equal to the preferred temperature, then the decrease will be no greater.

```

1.    // Updates the health of an organism
2.    private void updateHealth() {
3.        if (!stats.isDead()) { // If the organism isn't dead already
4.            double currentHealth = stats.getHealth(); // The current health
5.            double healthDecrease = 1.4; // Default decrease
6.            double temperatureDecrease = Math.abs(1 - stats.getTempFactor()); // The decrease
due to temperature factor

```

133

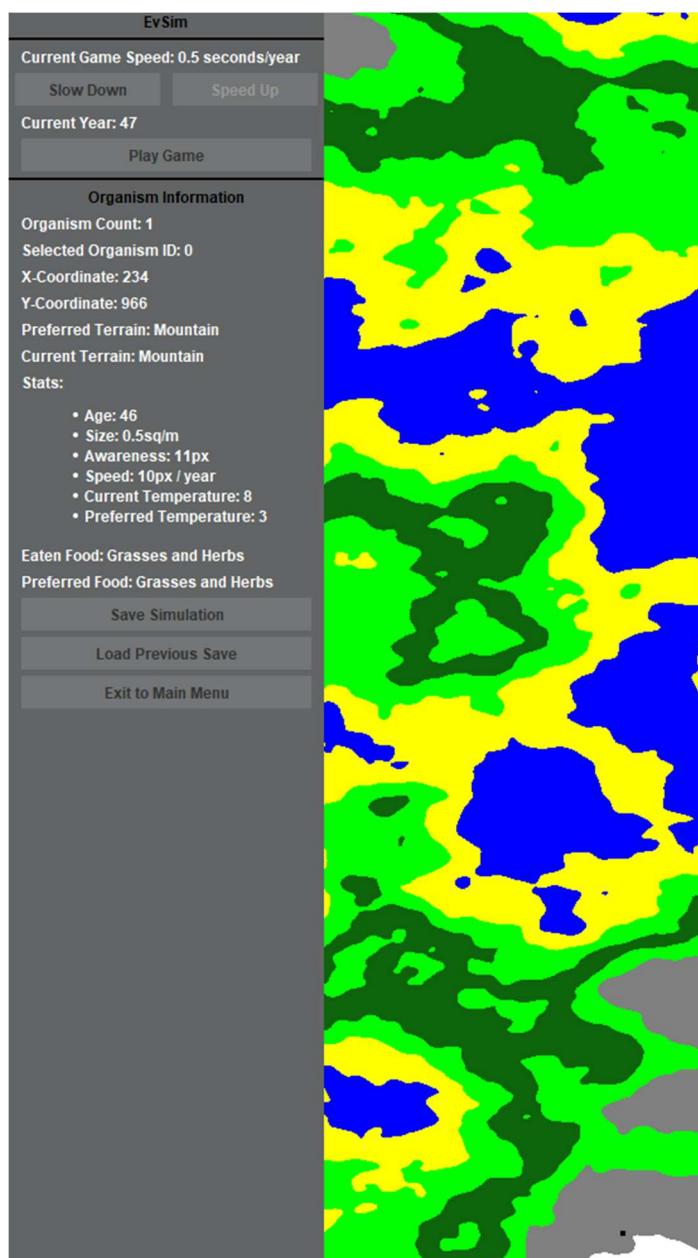
```

7.         double foodDecrease = Math.abs(1 - stats.getFoodFactor()); // The decrease due to
food factor
8.         double totalDecrease = healthDecrease + temperatureDecrease + foodDecrease; // The
total decrease amount
9.
10.        currentHealth = currentHealth - totalDecrease; // Decrease the health by the
decreasing amount
11.
12.        if (currentHealth <= 0) { // If the organism is dead, set it to be dead and then
round the health to 0
13.            stats.setDead(true);
14.            currentHealth = 0.0;
15.        } else if (currentHealth > 100)
16.            stats.setHealth(100.0); // If the health is ever greater than 100, cap it at
100
17.        stats.setHealth(Double.parseDouble(decimalFormat.format(currentHealth))); // Setting to 2 decimal places
18.    }
19.}

```

Java

As seen in the code block above, the health should now decrease by the amount explained above. The output (subsequent life span of the organism) of this section is shown below.



As seen in the screenshot to the right, the organism is dead (painted black at the bottom right). The organism was 46 years old at the time of death, and its preferred food was Grasses and Herbs. The preferred temperature was 3, with the current temperature of 8. From this, the expected time of death is shown below.

Grasses and herbs food factor = 0.25.
Subsequent decrease = 0.75.

Temperature factor (average) = 0.9.
Subsequent decrease = 0.2.

Total decrease = 2.35.
 $100 / 2.35 = 42.5$ years.

Considering the organism died on the 46th years old, this test can be considered a success, as with the health of the organism now being considered complete.

The organisms last by default 70 years, and the global factors influence this. The quickest an organism can die is ~35 years, giving the organism enough time to live a sufficient life. If the organisms are dying too fast, a feature can be added to extend the organisms life span, as the default factor can be changed from 1.4. Therefore, the health of an organism is now functional and correct, and also has structure which can be adapted to allow change by the user.

3.3.8 Survivability

The survivability will be a function which can be called during an organism's movement to check the organism's survivability within one area. This will be called frequently, so needs to be efficient and not demanding for the program to execute. Depending on how big the organism's awareness is (see 3.3.1), is how many spaces an organism can check before moving to. The survivability will be a prediction of how likely the organism is to survive in one area, for example if it is not the organism's preferred terrain then the survivability will be predicted significantly lower, or if the organism is not able to breed (see 3.4), or is a prey organism (see 3.5), the survivability will also decrease. Ideally, the organism will move to the area which has the highest survivability and therefore make a calculated decision before it moves.

3.3.8.1 Logic

The logic here is not too large, as it is only considering whether there is another organism near that coordinate, or if it is the preferred terrain. If the area is not one of the organism's preferred terrains, it is highly unlikely that that terrain will be beneficial for the organism, and therefore it will see a significant decrease in the survivability. Furthermore, if the organism is not looking to breed (will be implemented in 3.4), or the organism is not a predator (will be implemented in 3.5), then the survivability will also decrease. The survivability will also have a chance to be inaccurate (organism misjudgement), meaning that the organism may move to an area which is not the best for it.

The values for survivability will be set at default to 1, with other factors decreasing that. If the terrain the organism wants to move to for example is the second preferred terrain, then the survivability is 0.8, or if it is the third preferred terrain it is 0.6. Anything else will be 0.4. This is only a prediction which is made by the organism, therefore does not need to be accurate. Once the breeding and predator/prey cycles are complete, the survivability will also be affected, however this cannot be implemented until then.

3.3.8.2 Implementation

This will work alongside 3.4 Organism Movement, as the organism will move to the coordinate which has the highest survivability. See 3.4 Organism Movement for the use case of this function.

As mentioned in 3.3.8.1 Logic, this function is simple at this stage due to it only checking the terrain type. Shown below is the code block for the logic mentioned above.

```
1. // Gets the survivability of an organism at the coordinate position newX, newY
2. public double getSurvivability(int newX, int newY){
3.     double survivability = 1.0; // Default value
4.     double chanceToMisjudge = random.nextDouble(0, 1); // Small chance to make a mistake
5.
6.     String newTerrain = generateMap.getTerrain(newX, newY); // Gets the terrain at position
newX, newY
7.
8.     if(newTerrain.equals(preferredTerrain) && chanceToMisjudge > 0.05) { // If it is the
preferred terrain
9.         return survivability;
10.    } else if (newTerrain.equals(stats.getSecondPreferredTerrain()) && chanceToMisjudge >
0.05) { // If it is the second most preferred terrain
11.        return survivability - 0.2;
12.    } else if (newTerrain.equals(stats.getThirdPreferredTerrain()) && chanceToMisjudge >
0.05) { // If it is the third most preferred terrain
13.        return survivability - 0.4;
14.    } else return survivability - 0.6; // Else return 0.4
15. }
```

Java

This section will be expanded once breeding and predator prey cycles are implemented, for now, all that is needed to be returned is the survivability factor based off terrain.

Testing for this section will be carried out once the movement is improved. See 3.4 below.

3.4 Organism Movement

The movement of organisms will use the *survivability* (see 3.3.8) of the terrains around. The area in which an organism can move to, is based off the speed of the organism. If the speed is 10 pixels, then the organism can move to any space within a radius of 10 (x and y).

3.4.1 Logic

For this, multiple lists will be created, 2 storing the potential x and y coordinates, one storing the survivability's of each of those positions, and a final one storing the indexes of the highest survivability. The list of indexes will be then used to select at random a new x and y coordinate based off the highest survivability.

The lists of x and y coordinates will be created, adding in the available coordinates, so that another list – list of survivability - can be created. The list of survivability will contain the survivability of each x and y coordinates. Once this is created, a list of the highest survivability indexes can be created – highest survivability positions. One index can then be chosen at random from this list, which is where the organism will move to. The flowchart below represents this process visually.



3.4.2 Code

Below is the complete method for the organisms' new coordinates. Refer to comments for explanations of specific sections.

```
1. // The calculated decision made by each organism to move to the best position based on
2. // survivability
3. private void makeDecision() {
4.     // Lists which will contain the x and y coordinates which can be moved to
5.     List<Integer> xPositions = new ArrayList<>();
6.     List<Integer> yPositions = new ArrayList<>();
7.     // Populates the lists created above with the appropriate coordinates
8.     for(int counter = (int) (0 - organismSpeed); counter < organismSpeed; counter++){
9.         xPositions.add((int) (counter + organismSpeed), currentX + counter);
10.        yPositions.add((int) (counter + organismSpeed), currentY + counter);
11.    }
12.    // List of survivability levels for each x and y position
13.    List<Double> listOfSurvivability = new ArrayList<>();
14.    for(int counter2 = 0; counter2 < xPositions.size(); counter2++)
15.        listOfSurvivability.add(getSurvivability(xPositions.get(counter2), yPositions.get(counter2)));
16.
17.    // Find positions with the highest survivability
18.    double maxSurvivability =
19.        listOfSurvivability.stream().mapToDouble(Double::doubleValue).max().orElse(0); // Find the highest
20.        // survivability value in the list using stream
21.        List<Integer> highestSurvivabilityPositions = new ArrayList<>();
22.        // If the survivability is equal to the highest survivability, add the index to the
23.        // list of ideal positions
24.        for (int counter3 = 0; counter3 < listOfSurvivability.size(); counter3++) if
25.            (listOfSurvivability.get(counter3) == maxSurvivability)
26.                highestSurvivabilityPositions.add(counter3);
27.
28.    // Choose a random position of the list, this translates to the position of the x and y
29.    // coordinates
30.    int randomValue =
31.        highestSurvivabilityPositions.get(random.nextInt(highestSurvivabilityPositions.size()));
32.    currentX = xPositions.get(randomValue); // The x and y coordinates of the chosen
33.    position are now set and updated
```

```

26.         currentY = yPositions.get(randomValue);
27.
28.         // This ensures that the new X and Y coordinates are not outside the bounds of the
29.         // window
30.         // The x and y coordinate are compared against the width and height of the gameWindow,
31.         if higher or lower,
32.             // the coordinates are adjusted in the appropriate direction
33.             if (currentX >= GameWindow.WIDTH - 260) {
34.                 currentX = GameWindow.WIDTH - 300;
35.             } else if (currentX < 0) currentX = 10;
36.             if (currentY >= GameWindow.HEIGHT - 35) {
37.                 currentY = GameWindow.HEIGHT - 75;
38.             } else if (currentY < 0) currentY = 10;
39.         }

```

Java

3.4.3 Testing

To test this, I will track one organism's movement, checking that the correct position is moved to. If the organism moves to the position of the highest survivability, then this section is correct.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=49990:C:\Program Files\JetBrains
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\
IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\
lib\jackson-core-2.13.3.jar Main
2 X Coordinates: [109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119]
3 Y Coordinates: [222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232]
4 Survivability list: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.4, 1.0, 1.0, 1.0, 1.0]
5 Highest survivability positions: [0, 1, 2, 3, 4, 5, 7, 8, 9, 10]
6 New X Coordinate: 119
7 New Y Coordiante: 232

```

From this console result, we can analyse if this section is correct.

Printed on lines 2 and 3 are the X and Y coordinates which the organism can move to. The fourth line is the list of survivability. On here we can see 9 of the coordinates are the preferred terrain, and one is not (in this case, the 0.4 is the product of random chance to make a misjudgement). As seen on line 5, the highest survivability positions count all but the 0.4 in the list. The new coordinates are now chosen at random, and as seen from lines 6 and 7, the new coordinates are from the list, and from the correct positions.

Evidently, the expected result has taken place, and this section can be considered correct.

Considering how the movement is now dependant on the survivability defined in 3.3.8, to increase the accuracy or change the calculations of the movement, the survivability class is the one which needs to be changed. This is being noted due to expansion of this being needed once organism breeding and predator prey cycles are implemented. See later sections for the improvements made to movement and survivability calculations.

3.4.4 Testing survivability

3.4.4.1 Error

We can also gauge whether the survivability calculations are correct from this console log. The organism in this simulation was in a water biome (see image to the right), meaning each coordinate should be 1 (preferred is water), and as seen from line 4, this is correct. However, the 0.4, which has been created because of random misjudgement will never influence the organism. Due to the misjudgement removing survivability, the only effect this has is the organism not moving to that position, instead of an increased chance to go to the position.

3.4.4.2 Improvement to the survivability

To correct this error, instead of checking the chance to misjudge during each for statement (lines 8, 10, 12 in 3.3.8.2), I will add a line before (line 7), as follows.

```
1.     if(chanceToMisjudge < 0.05) return survivability; // Random chance to misjudge
```

Java

This will mean that instead of decreasing the survivability, the survivability has a chance to return at full, increasing the chance of a meaningful misjudgement.

3.5 Organism Breeding

This section will not only combine two organism's attributes (breeding), but also give organisms evolutionary abilities and adaptations.

3.5.1 Breeding

For organisms to breed, both will have to be over the age of 10 and be within a radius of 5. On spawn, organisms will be assigned a male and female attribute, organisms which follow the above criteria and are opposite sex, will breed to spawn a child organism. The child organism will inherit attributes from both parents, with a bias to each side. For example, the child organism may take the mothers awareness, and the fathers speed, or both from the same parent. This is much like in real life where attributes (especially visual) are followed along through genes.

This means that the organisms will have to have an attribute called sex, which will be assigned to male, female, or asexual (see 3.5.1.1). On each year, after all the organisms have moved and updated their stats, the organisms will be checked for the above criteria (within 5 pixels, male and female), and if it matches, then the breeding will take place. This will add in a new organism into the JSON, which stores the IDs of the parents as well as the shared attributes. As covered within many sections of 3.3 Organism Statistics, some of the attributes which are taken from the parents will become greater throughout generations, such as the awareness and size. This may require small changes to each of the stat methods.

3.5.1.1 Breeding Criteria

First was to ensure that the breeding criteria is met, checking their sex, age, and position. If these are all correct, then the process of a new organism can take place.

```
1. // This will contain both breeding cycle calculations and predator / prey cycle
2. calculations
3. private void newOrganismEncounter(int originOrganismId, int secondaryOrganismId){
4.     Organism originOrganism = organisms[originOrganismId]; // The two organisms are fetched
5.     Organism secondaryOrganism = organisms[secondaryOrganismId];
6.
7.     if(originOrganism.getSex() != secondaryOrganism.getSex()){// If they are male and
8.         female
9.         if(originOrganism.getStats().getAge() >= 10 &&
secondaryOrganism.getStats().getAge() >= 10){ // If both are older than 15
10.             if(Math.abs(originOrganism.getxCoordinate() -
secondaryOrganism.getxCoordinate()) <= 5 && Math.abs(originOrganism.getyCoordinate() -
secondaryOrganism.getyCoordinate()) <= 5) { // If they are within a radius of 5
11.                 if(originOrganism.getSex() == 1){ // To find the father and mother
12.                     createChildOrganism(originOrganism, secondaryOrganism);
13.                 } else {
14.                     createChildOrganism(secondaryOrganism, originOrganism);
15.                 }
16.             }
17.         }
}
```

Java

Within this code block, the above criteria are checked for organism breeding. If the criteria is met, then the `createChildOrganism` method is called, which will create a new organism, taking attributes from both the mother and father.

3.5.1.2 Increasing the breeding chance

One issue which was presented within this code block, was the fact that the organisms die before a sufficient new generation of organisms are created. For example, in a simulation which ran for 66 years, there was enough time for one new child organism to be created, before the organisms became extinct. To resolve this, I will attempt two options: A) increase the life span of organisms, and B) spawn the organisms closer to each other.

Increasing the life span of the organisms will give the organisms more chance to breed, however creating the organisms in a smaller area means that (especially for low a low organism count) organisms are much more likely to find another organism, compared to random spawning organisms that are halfway across the map.

For option A, the health decrease each year was changed to 0.8, meaning on average organisms survived for 60 years compared to 40, giving a much larger amount of time to live and breed before dying.

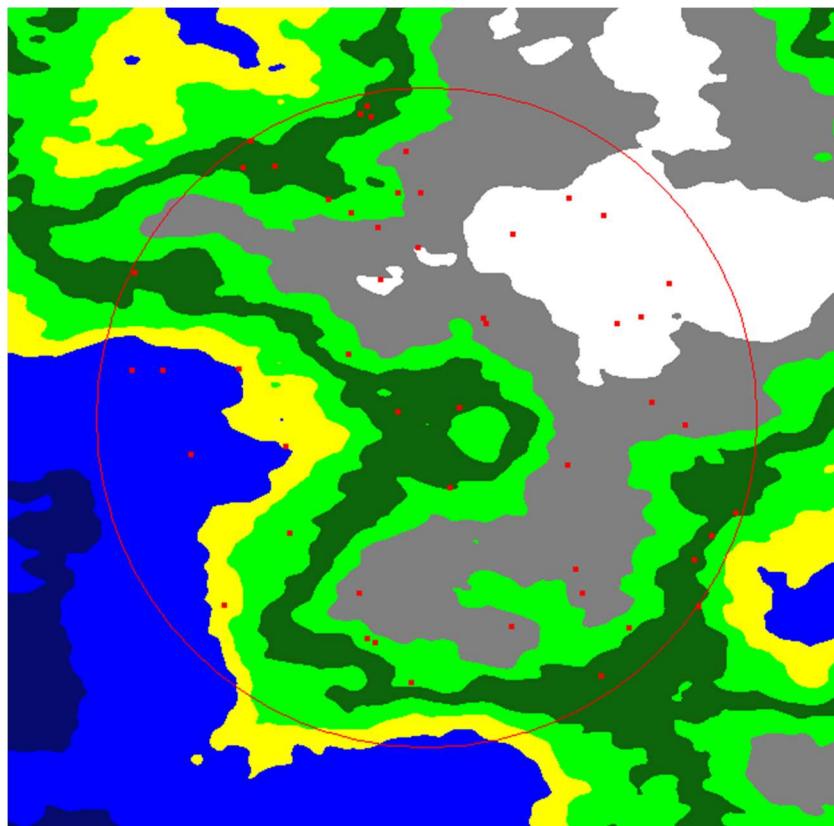


Figure 3.5.2.2 a

For option B, when the new organisms are generated, they will generate in a denser radius at the centre of the map. This will be around an epicentre ($\text{width and height} / 2$), with the radius equal to the amount of organisms * 5. This means if there are 50 organisms, they will spawn in a circle size (perimeter) 1570 pixels. See figure 3.5.2.1.1a, which represents this.

Having population density at the start of the simulation will significantly increase the chance of organisms breeding. Testing this, the number of organisms breeding is now sufficient, with around 35% reproducing, and the oldest organism dying at age 90. The code for the new organism location is shown below.

```

1.     private void newOrganismLocation(int Width, int Height) {
2.         int epicenterX = Width / 2; // Calculate where the centre of the epicentre should be
(middle of the map)
3.         int epicenterY = Height / 2;
4.
5.         // Set the epicenter radius to be at most one-fourth of the map width or preferably 5
times the number of organisms
6.         int epicenterRadius = Math.min(Width / 4, ORGANISM_COUNT * 5);

```

```

7.      // Generate random coordinates for each organism
8.      for (int i = 0; i < ORGANISM_COUNT; i++) {
9.          // Generate a random angle in radians
10.         double angle = 2 * Math.PI * random.nextDouble();
11.
12.         // Generate a random distance within the radius
13.         double distance = epicenterRadius * Math.sqrt(random.nextDouble());
14.
15.         // Calculate the x and y coordinates for the organism based on the polar
16.         // coordinates
17.         // Convert polar coordinates (angle and distance ^) to cartesian coordinates (x
18.         // and y - what this program uses)
19.         int organismX = (int) (epicenterX + distance * Math.cos(angle));
20.         int organismY = (int) (epicenterY + distance * Math.sin(angle));
21.
22.         // Ensure the organism is within the map boundaries and sets to global variables x
23.         // and y
24.         x = Math.max(0, Math.min(Width - 1, organismX));
25.         y = Math.max(0, Math.min(Height - 1, organismY));
26.     }

```

Java

3.5.1.3 Creating the new organism

This section covers the creation of the new organism within the JSON file, which will take attributes from both the mother and the father equally.

Within the new organism encounter method in 3.5.2.1, there is another method called, `createChildOrganism`. This is where the new organism will be added into the JSON, with its stats inheriting from both the mother and the father. For this, a return to json-simple will be required, as I will use the same method from `generateOrganisms` to add a new organism to the JSON. Due to how the JSON is read and updated throughout the program, the organisms cannot be added the instant that organisms meet, instead being added after all organisms have been updated. This means that any new organisms created will be added to a list, with the list being added to the JSON once all organisms are updated.

As mentioned, it will follow a similar structure to how the organisms are generated at the start. Shown below, the method creates a new JSON Object, and adds it to the list of organisms to add at the end.

```

1.  private void createChildOrganism(Organism father, Organism mother) {
2.      // Create a new organism as a JSON object
3.      JSONObject newOrganism = new JSONObject(); // Creates a new JSON object for each
organism
4.
5.      // Adds a key/value pair to the json object initialised above
6.      newOrganism.put("organismId", organisms.length);
7.      newOrganism.put("xCoordinate", mother.getxCoordinate()); // Spawns at its mother
8.      newOrganism.put("yCoordinate", mother.getyCoordinate());
9.      newOrganism.put("currentTerrain", generateMap.getTerrain(mother.getxCoordinate(),
mother.getyCoordinate()));
10.
11.     JSONObject statsObject = getStatsObject(father, mother); // Create the stats, will need
to take attributes from mother and father
12.
13.     newOrganism.put("stats", statsObject);
14.     newOrganism.put("sex", random.nextInt(0, 2)); // Random between male and female
15.     newOrganism.put("fatherID", father.getOrganismId());
16.     newOrganism.put("motherID", mother.getOrganismId());
17.
18.     organismsToAdd.add(newOrganism); // Adding to the new organisms to be saved at the end
of the year
19.  }

```

Java

The list that the new organism is added to will then be added into the JSON once all organisms are updated, as seen in the code block below.

```
1.    // This method ensures that the new organisms are added at the end of the year (when
2.    // organisms have finished updating)
3.    // so that there are no discrepancies between GameWindow's organisms and the JSON file
4.    private void addNewOrganisms(List<JSONObject> newOrganisms){
5.        JSONArray jsonArray = new JSONArray(); // This will contain the current JSON array
6.        JSONParser parser = new JSONParser();
7.
8.        try (FileReader reader = new FileReader("organisms.json")) {
9.            // Parse the JSON file
10.           Object obj = parser.parse(reader);
11.           jsonArray = (JSONArray) obj;
12.
13.           jsonArray.addAll(newOrganisms); // Adding in the new organisms from the
14.           // organismsToAdd list
15.           // Save the JSON array to the file
16.           FileWriter fileWriter = new FileWriter("organisms.json");
17.           fileWriter.write(jsonArray.toJSONString());
18.           fileWriter.flush();
19.       } catch (IOException | ParseException e) {
20.           e.printStackTrace();
21.       }
22.   }
```

Java

To test this, I will run a simulation of 50 organisms. If the organism's ID is higher than 49 (meaning it is a new organism), it will be painted to the screen as magenta.

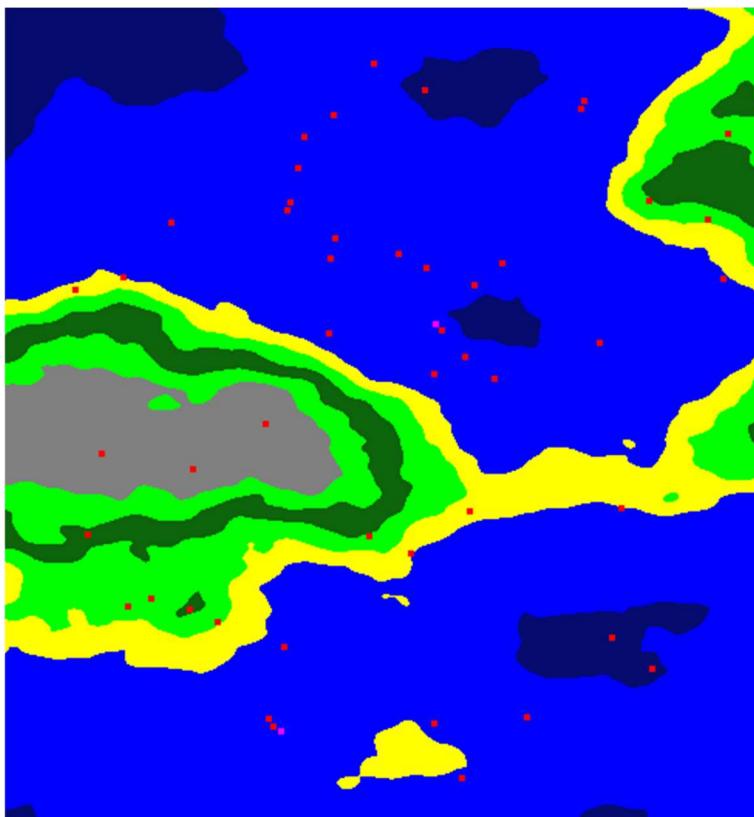


Figure 3.5.2.3 a

Shown in figure 3.5.2.3a above, there are two magenta organisms on here, showing that 2 organisms have been created. This screenshot was taken from year 15, figure 3.5.2.3b below shows the same simulation, but 100 years later.

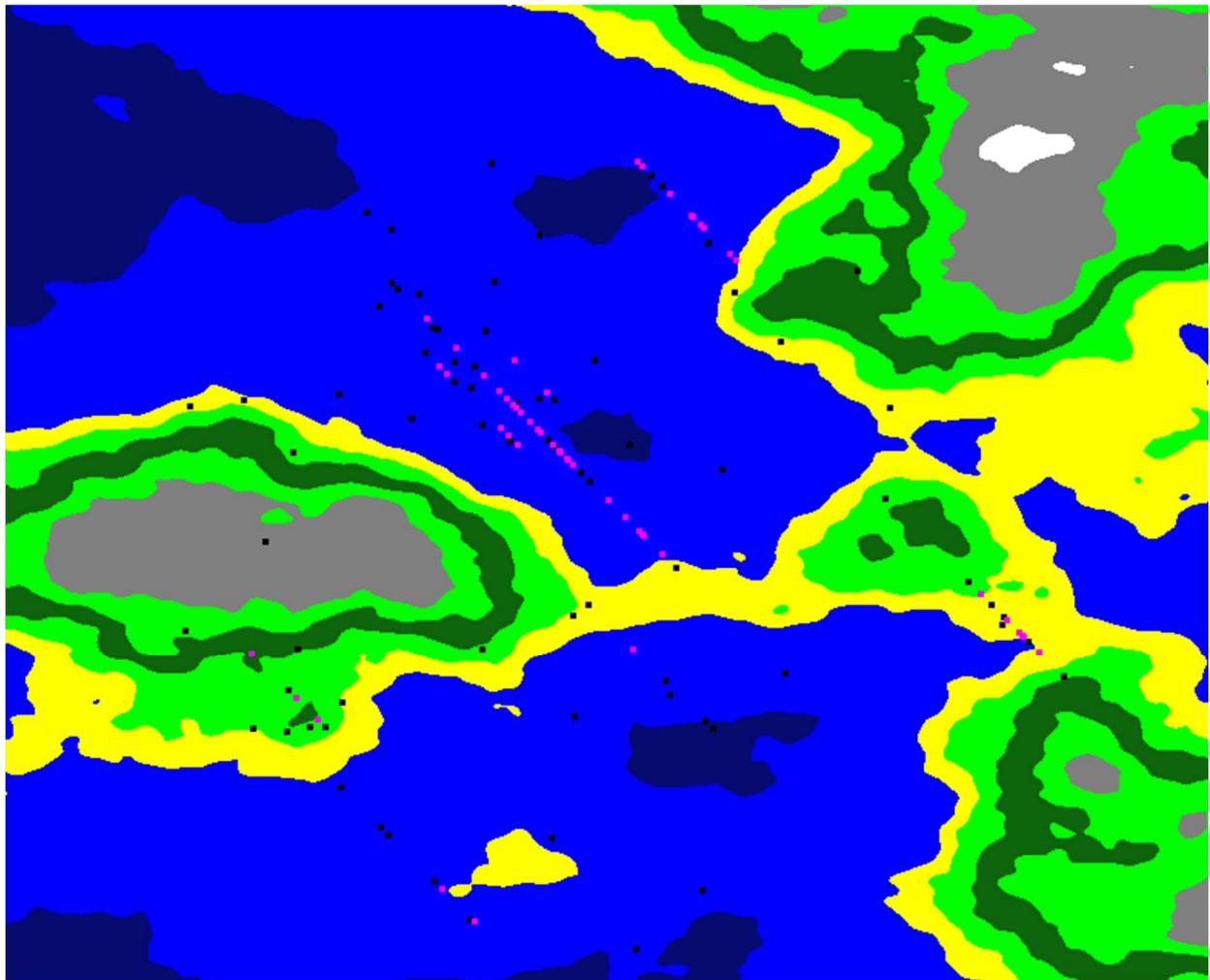


Figure 3.5.2.3 b

In figure 3.5.2.3b, it is clear that organisms are able to breed, with the organisms in black being dead, and the organisms in magenta being a new organism. However, it is also clear from this that the movement is diagonally biased, with a line of organisms stretching across the ocean, and the organisms breeding very frequently. Organism movement at this stage is only impacted by the terrain, this will change significantly once predator prey cycles are implemented. Due to the predator prey cycles implementing the feature for organisms to move towards and away from each other, it would be inefficient to implement this here for breeding, when it will be changed once predator prey cycles are present. Due to this section's aim being to implement breeding, I am going to leave the movement for 3.6 Predator / Prey Cycles, avoiding redeveloping code.

3.5.1.4 Taking attributes from the mother and father

While the organisms do now breed and take some attributes (such as parental ID), the stats are not taken from the parents. Currently, each organism that is new, is (attribute wise) the same as each organism at the beginning of the simulation, with very basic stats. The aim of this section is to allow the organisms to take attributes from the parent organisms, in a way that they would in real life (taking some attributes from the mother and some from the father). In the code block below, each of the attributes are updated, and set to be representative of either one parent or a combination of both with added variation. Refer to the comments, which explain how the organism takes its parents attributes.

3.5.1.4.1 Implementation

```

1.    // Sets the new stats to be representative of parental attributes
2.    @SuppressWarnings("unchecked")
3.    private JSONObject getStatsObject(Organism father, Organism mother) {
4.        JSONObject statsObject = new JSONObject(); // The stats object is initialised
5.
6.        double randomVariation = random.nextDouble(-0.2, 0.2); // Random variation to either
side
7.        int randomVariation2 = random.nextInt(-2, 2); // Random integer variation to either
side
8.        double chooseParent = random.nextInt(0, 2); // Choose to take from the mother or father
9.
10.       // Random awareness from average of both parents with added double variation
11.       double newAwareness = (father.getStats().getCurrentAwareness() +
mother.getStats().getCurrentAwareness()) / 2.0;
12.       statsObject.put("currentAwareness", newAwareness + randomVariation);
13.
14.       statsObject.put("organismsEncountered", 0.0); // Updated during execution of program
15.       statsObject.put("awarenessHistory", new JSONArray()); // Only used during first 5
years, default value
16.       statsObject.put("age", 0); // The age needs to be representative and therefore start at
0
17.
18.       FoodTypes newFood; // Chooses to eat food which either parent eats
19.       if(chooseParent == 1) newFood = father.getStats().getPreferredFood();
20.       else newFood = mother.getStats().getPreferredFood();
21.       statsObject.put("preferredFood", newFood);
22.
23.       statsObject.put("eatenFood", null); // Has not eaten yet, so does not need to be set
24.       statsObject.put("recentFood", new JSONArray()); // Used during first few years of
organisms life
25.       statsObject.put("foodFactor", 0.0); // Calculated after an organism eats
26.       statsObject.put("sizeFactor", 0.0); // Calculated after an organism eats
27.
28.       // Size is taken from father or mother with added double variation
29.       double newSize = (father.getStats().getSize() + mother.getStats().getSize()) / 2;
statsObject.put("size", newSize + randomVariation);
30.
31.       // Speed is taken from father or mother with added double variation
32.       double newSpeed = (father.getStats().getSpeed() + mother.getStats().getSpeed()) / 2;
statsObject.put("speed", newSpeed + randomVariation);
33.
34.       statsObject.put("currentTemp", 0); // Changes during the organisms life, no default
value
35.
36.       int newTemp; // Temp is taken from either father or mother, with an added integer
variation
37.       if(chooseParent == 1) newTemp = father.getStats().getPreferredTemp();
38.       else newTemp = mother.getStats().getPreferredTemp();
39.       statsObject.put("preferredTemp", newTemp + randomVariation2);
40.
41.       statsObject.put("recentTemp", new JSONArray()); // Changes during organisms life, no
default value
42.       statsObject.put("tempFactor", 1); // Changes during the organisms life, no default
value
43.
44.       String newTerrain; // Preferred terrain is taken from either father or mother
45.       if(chooseParent == 1) newTerrain = father.getStats().getPreferredTerrain();
46.       else newTerrain = mother.getStats().getPreferredTerrain();
47.       statsObject.put("preferredTerrain", newTerrain);
48.
49.       statsObject.put("secondPreferredTerrain", "Default"); // Will be chosen by the organism
50.       statsObject.put("thirdPreferredTerrain", "Default"); // Will be chosen by the organism
51.
52.       statsObject.put("health", 100.0); // Default value for all organisms
53.       statsObject.put("dead", false); // Default value for all organisms
54.       statsObject.put("ageLastBred", -1); // Default value for all organisms
55.
56.       return statsObject;
57.    }

```

Java

The only other factor which should change after this, is checking whether the organism already has factors such as a preferred food and temperature when they are set at first. For example, the organisms at the beginning will always set their preferred food as it is expected to be set as null, however now that organisms take from the parents, this is not true. Below is an example of how this is avoided.

```

1. if(stats.getAge() < 5 && stats.getPreferredFood() == null){
2. /* Other code */
3. } else if (stats.getAge() == 5 && stats.getPreferredFood() == null) {
4. /* Other code */

```

In this, it is checked beforehand whether the preferred food is already set, so that it is not replaced, and does not waste time adding recently eaten food.

3.5.1.4.2 Error

However, running the program at this stage returned an error. It appears that JSON Simple cannot save the preferred food as a class, unlike Jackson does. The console below shows the error which was returned, alongside the JSON which was generated.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=60343:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 {"xCoordinate":852,"motherID":41,"organismID":50,"yCoordinate":639,"stats":{}
  "thirdPreferredTerrain":"Default","preferredTerrain":"Water","awarenessHistory":[],"health":100.0,"dead":false,"preferredFood":FoodTypes.FoodTypes@52ebb59b,"secondPreferredTerrain":"Default","ageLastBred":-1,"recentFood":[],"speed":4.137413295455939,"preferredTemp":10,"eatenFood":null,"recentTemp":[],"foodFactor":0.0,"size":0.4624132954559395,"organismsEncountered":0.0,"tempFactor":1,"currentTemp":0,"sizeFactor":0.0,"age":0,"currentAwareness":1.5624132954559395},"sex":0,"currentTerrain":"Water","fatherID":47}
3 com.fasterxml.jackson.databind.JsonMappingException: Unrecognized token 'FoodTypes': was expecting (JSON String, Number, Array, Object or token 'null', 'true' or 'false')
4 at [Source: (File); line: 1, column: 45882] (through reference chain: java.lang.Object[][],[50]->Organism["stats"])

```

The highlighted sections here are what to focus on. Jackson cannot convert “FoodTypes.FoodTypes@52ebb59b” into the class of FoodTypes. This means that in order to set the organisms preferred food at the start of its life, I will have to add in some form a string name of the preferred food, and have it converted to FoodTypes and saved once the program is handling JSON through Jackson. The stats for the organism are now created as follows:

```

1.     statsObject.put("preferredFood", null); // Has to be set as FoodTypes however json simple cannot set as type class. See below
2.
3.     FoodTypes newFood; // Chooses to eat food which either parent eats
4.     if(chooseParent == 1) newFood = father.getStats().getPreferredFood();
5.     else newFood = mother.getStats().getPreferredFood();
6.     statsObject.put("preferredFoodString", newFood); // Sets as a string to be changed to a FoodType

```

Java

With this being converted during updateFood method:

```

1.     if(stats.getPreferredFoodString() != null) {
2.         stats.setPreferredFood(getFoodType(stats.getPreferredFoodString()));
3.         stats.setPreferredFoodString(null);
4.     }

```

Java

Testing this returned no errors, with organisms being created as expected.

3.5.1.4.3 Organism generations

One final factor to add in is an organism's generation. For example, the organisms that spawn at the beginning will be first generation, and the organisms which are bred will be second generation. The program should have a way of storing this, therefore I have added in "generation" into the JSON and will be updated in the attributes which are taken.

```
1.      // The generation of the organism set to the highest of both parents
2.      int generation = Math.max(father.getStats().getGeneration(),
3.          mother.getStats().getGeneration());
4.      statsObject.put("generation", generation);
```

Java

3.5.1.4.4 Resultant organism example

This shows what a new organism may look like within the JSON.

```
1. {
2.     "xCoordinate": 723,
3.     "motherID": 28,
4.     "organismId": 50,
5.     "yCoordinate": 432,
6.     "stats": {
7.         "generation": 1,
8.         "thirdPreferredTerrain": "Default",
9.         "preferredTerrain": "Mountain",
10.        "awarenessHistory": [],
11.        "health": 100.0,
12.        "dead": false,
13.        "preferredFood": null,
14.        "secondPreferredTerrain": "Default",
15.        "ageLastBred": -1,
16.        "recentFood": [],
17.        "speed": 8.63057679144559,
18.        "preferredTemp": 7,
19.        "eatenFood": null,
20.        "recentTemp": [],
21.        "foodFactor": 0.0,
22.        "size": 0.4855767914455892,
23.        "organismsEncountered": 0.0,
24.        "tempFactor": 1,
25.        "preferredFoodString": "Coniferous Tree Spawn",
26.        "currentTemp": 0,
27.        "sizeFactor": 0.0,
28.        "age": 0,
29.        "currentAwareness": 1.265576791445589
30.    },
31.    "sex": 1,
32.    "currentTerrain": "Grass",
33.    "fatherID": 42
34. }
```

JSON

3.5.1.5 Testing the breeding cycle

For this section to be complete, organisms must breed if they are male and female, and are within a 5-metre radius. If one pair of organisms successfully breeds, both the parent's attributes and the new organism's attributes will be checked, if there is an equal proportion and balance between the two, this section can be considered a success. I will check over multiple generations of organisms as well.

3.5.1.5.1 First Generation Breeding

This section covers the testing of organisms taking attributes from the parents. In this simulation I will have 2 organisms, keeping the simulation running until these organisms' breed. Once they have bred, I will end the simulation and compare the JSON result. Below is the resultant JSON (some factors omitted to save space).

```

1. [
2.   {
3.     "organismId": 0,
4.     "stats": {
5.       "currentAwareness": 1.4,
6.       "organismsEncountered": 0.5,
7.       "awarenessHistory": [],
8.       "age": 19,
9.       "preferredFood": {},
10.      "preferredFoodString": "Aquatic life / Spawn",
11.      "eatenFood": {},
12.      "recentFood": [],
13.      "foodFactor": 0.1,
14.      "sizeFactor": 2.3,
15.      "size": 0.5,
16.      "speed": 6.96,
17.      "currentTemp": 1,
18.      "preferredTemp": 0,
19.      "recentTemp": [],
20.      "tempFactor": 0.96,
21.      "preferredTerrain": "Deep Water",
22.      "secondPreferredTerrain": "Default",
23.      "thirdPreferredTerrain": "Default",
24.      "health": 73.04,
25.      "dead": false,
26.      "ageLastBred": 15,
27.      "generation": 1
28.    },
29.    "currentTerrain": "Deep Water",
30.    "xCoordinate": 814,
31.    "yCoordinate": 524,
32.    "sex": 0,
33.    "fatherID": -1,
34.    "motherID": -1
35.  },
36.  {
37.    "organismId": 1,
38.    "stats": {
39.      "currentAwareness": 0.56,
40.      "organismsEncountered": 0.1,
41.      "awarenessHistory": [],
42.      "age": 19,
43.      "preferredFood": {},
44.      "preferredFoodString": "Plankton",
45.      "eatenFood": {},
46.      "recentFood": [],
47.      "foodFactor": 0.1,
48.      "sizeFactor": 3.0,
49.      "size": 0.5,
50.      "speed": 9.11,
51.      "currentTemp": 3,
52.      "preferredTemp": 1,
53.      "recentTemp": [],
54.      "tempFactor": 0.92,
55.      "preferredTerrain": "Deep Water",
56.      "secondPreferredTerrain": "Default",
57.      "thirdPreferredTerrain": "Default",
58.      "health": 73.9,
59.      "dead": false,
60.      "ageLastBred": 15,
61.      "generation": 1
62.    },
63.    "currentTerrain": "Deep Water",
64.    "xCoordinate": 833,
65.    "yCoordinate": 537,
66.    "sex": 1,
67.    "fatherID": -1,
68.    "motherID": -1
69.  },
70.  {
71.    "organismId": 2,

```

```

72.     "stats": {
73.         "currentAwareness": 1.1,
74.         "organismsEncountered": 0.4,
75.         "awarenessHistory": [],
76.         "age": 3,
77.         "preferredFood": {},
78.         "preferredFoodString": "Plankton",
79.         "eatenFood": {},
80.         "recentFood": [],
81.         "foodFactor": 0.1,
82.         "sizeFactor": 0.3,
83.         "size": 0.5289530091908712,
84.         "speed": 8.16,
85.         "currentTemp": 0,
86.         "preferredTemp": -1,
87.         "recentTemp": [],
88.         "tempFactor": 1.0,
89.         "preferredTerrain": "Deep Water",
90.         "secondPreferredTerrain": "Default",
91.         "thirdPreferredTerrain": "Default",
92.         "health": 100.0,
93.         "dead": false,
94.         "ageLastBred": -1,
95.         "generation": 1
96.     },
97.     "currentTerrain": "Deep Water",
98.     "xCoordinate": 811,
99.     "yCoordinate": 521,
100.    "sex": 0,
101.    "fatherID": 1,
102.    "motherID": 0
103. }
104. ]

```

JSON

In the above JSON, the child is organism ID 2, with the mother of organism ID 0, and father organism ID 1. Refer to those sections when father, mother, or child is mentioned.

Starting with the awareness:

The mother's awareness was 1.4, and the father's awareness was 0.56. The expected child awareness should be between the range of 0.78 and 1.18. With the child's awareness being 1.1, the awareness can be validated.

Preferred food:

Considering this is randomly chosen from its parents, the expected result should be either one of the parents preferred food. The mother's preferred food was Aquatic life / Spawn, and the father's preferred food was Plankton. The child's preferred food was Plankton, and therefore the preferred food is validated.

Size:

The size of the organism should be relative to the food factor of the organisms. During breeding, organisms should take the average of both parents, with an added variation. The father's size was 0.5, so was the mothers. The child's size was 0.5289530091905712. Considering the size can change quite significantly (bounds of 0.5), it would be more realistic to represent evolution of size through breeding (not varied) instead of through age. For this, I will remove parts of what was developed within 3.3.3 Size and transfer it over to when organisms breed. I will also limit this factor to only have 2 decimal places, to avoid long and unnecessary decimals. See 3.5.2.6 for this correction.

Speed:

The mother's speed was 6.96, and the fathers speed was 9.11. The child's speed was 8.16. The range that the speed should be is 7.8 to 8.2. It is clear therefore that the speed has been taken from both parents, and speed can be validated.

Preferred Temperature:

The mothers preferred temperature was 0, and the father's preferred temperature was 1. The child's preferred temperature was -1. The child's temperature should therefore be between the range of -2 and 3 (integer rounding). The preferred temperature can therefore be validated.

Preferred Terrain:

Both parents preferred terrains were Deep Water, and they had not visited any other terrains, therefore for the child's preferred terrain to be Deep Water as well, validates this selection as well.

Generation:

Considering the simulation, the only generation this organism should be, is 2. However, the JSON displays the child organism as being the first generation. The error here is extremely simple, as the program only takes the highest of the parents generations, instead of taking the highest and adding on 1. The corrected code for organism generation is shown below.

```
1.      // The generation of the organism set to the highest of both parents + 1
2.      int generation = Math.max(father.getStats().getGeneration(),
mother.getStats().getGeneration());
3.      statsObject.put("generation", generation + 1);
```

Java

3.5.1.5.2 Later Generation Breeding

This section covers organism evolution over an extended period. For this test, a simulation of 250 organisms was performed until all organisms became extinct. Below are the results of this. Figure 3.5.2.5.2a shows the global statistics displayed on the side menu, figure 3.5.2.5.2b shows an example of one organism.

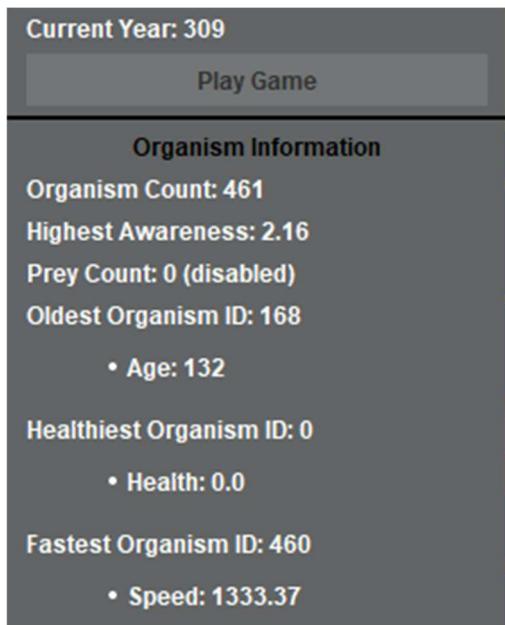


Figure 3.5.2.5.2 a

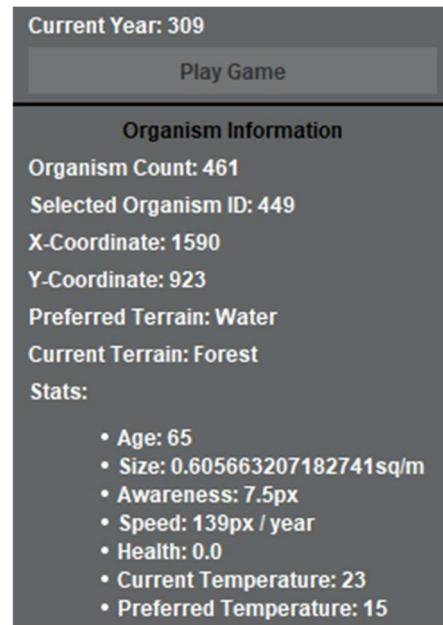


Figure 3.5.2.5.2 b

One of the most prominent issues in both Figure 3.5.2.5.2a and 3.5.2.5.2b is the speed of the organisms. The fastest organism throughout the 309 years had a speed of 1333.37, a speed which is enough to cover the whole map in one year. The main cause for this is the organisms taking the average of its parents speeds, meaning growth is unrestricted. If a dampening effect was to be added in, it would take a significant amount of effort for an organism to reach levels of 50 px / year, nevermind 1300+. The fix for this is going to be within 3.5.2.7, so refer there for the fix to this error.

Another issue prominent in 3.5.2.5.2b is the size of an organism, however with this being noted in 3.5.2.5.1 already, refer to 3.5.2.6 for the fix. These are the only errors present during testing of the evolution process and therefore once these are corrected, evolution at this stage can be validated.

3.5.1.6 Changing Size

As mentioned within 3.5.2.5.1, the size of the organisms will evolve through the children of the organisms instead of when they reach a certain age. This will require a change in both logic and implementation. The original logic uses a size factor to determine the organism's size change, I will follow a similar structure here, however considering the organisms can be of any age, I cannot compare it to a static figure. Instead, the size will increase if the average size factor of both parents is above 0.25 and decrease if it is below.

This means that the updateSize method can be minimised down to the following code.

```
1. // Update the organisms size
2. private void updateSize(){
3.     stats.setSizeFactor(stats.getSizeFactor() + stats.getEatenFood().foodFactor); // Update
the total food factor
4. }
```

Java

Considering this method is only one line, it can be moved into the updateStats method instead of a method by itself.

Below are the calculations which go towards the new size for the generated organism.

```
1. // An average size factor is calculated for both mother and father
2. double averageFatherSizeFactor = father.getStats().getSizeFactor() / stats.getAge();
3. double averageMotherSizeFactor = mother.getStats().getSizeFactor() / stats.getAge();
4. double newSize = 0;
5. if(averageFatherSizeFactor >= 0.25) newSize = newSize + 0.05; // These contribute to
the increase or decrease in size for the new organism
6. else newSize = newSize - 0.05;
7. if(averageMotherSizeFactor >= 0.25) newSize = newSize + 0.05;
8. else newSize = newSize - 0.05;
9. if(chooseParent == 1) newSize = newSize + father.getStats().getSize(); // Then chooses
a random parent to take size from
10. else newSize = newSize + mother.getStats().getSize();
11. statsObject.put("size", Double.parseDouble(decimalFormat.format(newSize))); // Sets to
2dp
```

Java

I will now conduct the same tests as 3.5.2.5 to ensure the size of the organisms change as intended over generations.

This simulation replicates what is found in 3.5.2.5.1, however the focus here is the organism's size. The result of this simulation is as follows.

The father's size factor was 3.8, and the mother's size factor was 3.7. Both organisms are 17 years old, meaning the average size factor for the father is 0.22, and the mothers average size factor being 0.21. This means that the organism's size should be 0.4, and as shown in the JSON (omitted stats) below, it is correct.

```
1. {
2.     "xCoordinate": 980,
3.     "motherID": 40,
4.     "organismID": 50,
5.     "yCoordinate": 358,
6.     "stats": {
7.         "size": 0.4
8.     },
9.     "fatherID": 18
10. }
```

JSON

This simulation replicates what is found in 3.5.2.5.2, however once again the size is of focus. The result of the simulation is shown below.

After 259 years, the last remaining organism had a size of 0.8, and was the sixth generation. This means the organism's ancestors would have had a varied history of size factors, increasing on average 0.05 a year. This shows that the organisms now develop their size more reliably, and over generations, instead of throughout their lifetime.

3.5.1.7 Changing Speed

As mentioned, the reason for the exponential growth of organisms speed, is the fact that they are inheriting the average of the parents, meaning there is no limit, and it can also never decrease. Instead of taking the average of both parents, the new speed will be based off the default value of 5. If the parents of the organism have a speed greater than 10 upon giving birth, the child's speed is set to the starting speed of (one of) the parent's speed + 0.5, and if it is lower than 10, -0.5. This means that the slower organisms are when giving birth, the slower the child organism will start off. This would be representative of evolution, as the organisms inherit both the organisms current and historical state.

Below is the adapted code for the speed of a new organism.

```
1.      // Takes the parents starting speed, and decreases if the current speed is low, and
visa-versa
2.      double newSpeed;
3.      if(chooseParent == 1) newSpeed = father.getStats().getStartSpeed();
4.      else newSpeed = mother.getStats().getStartSpeed();
5.      if(mother.getStats().getSpeed() > 10.0) newSpeed = newSpeed + 0.5;
6.      else newSpeed = newSpeed - 0.5;
7.      statsObject.put("speed", Math.min(30.0, Math.max(3.0, newSpeed))); // Lowest it can be
is 3, highest is 30.
```

Java

Testing this, it is clear from the screenshot below that organism's speed is now limited.

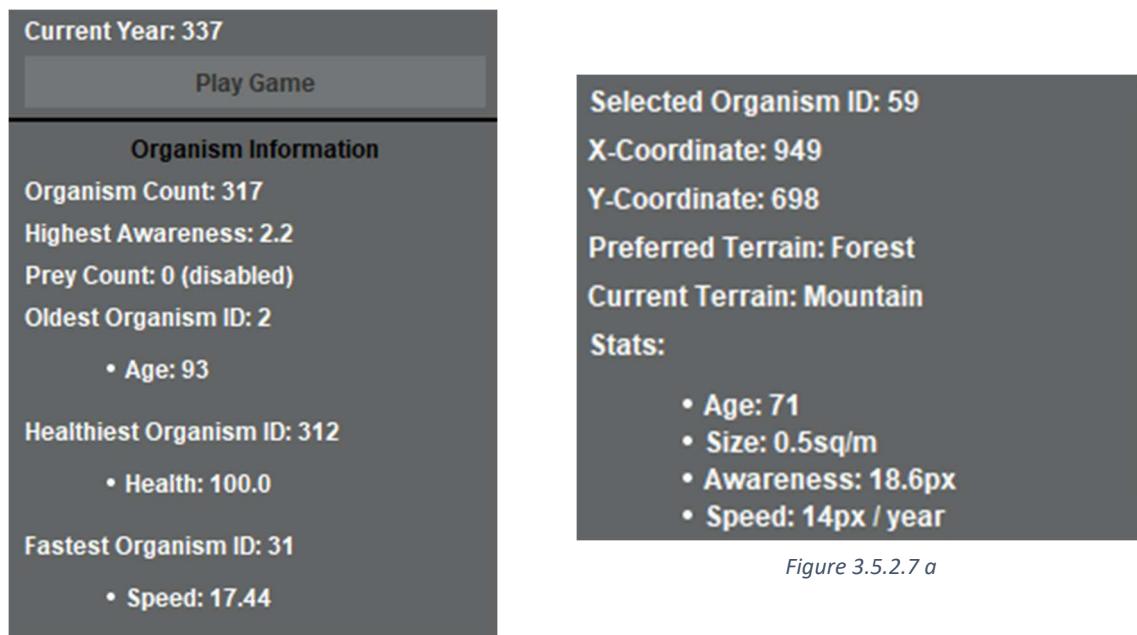


Figure 3.5.2.7 a

Figure 3.5.2.7 b

Figure 3.5.2.7a shows a second generation organism which has since died. Its ending speed was 14, an appropriate speed for a 70 year old organism, backed up with the table displayed in 3.3.4. Figure 3.5.2.7b shows the fastest organism throughout the 337 years to have a speed of 17.44. This shows that organisms no longer get faster by an exponential amount, with the organism here being a first generation organism. This now confirms that the breeding is correct.

3.5.2 Evolutionary adaptations

This section will cover the requirements for organisms over generations to have evolutionary adaptations. This section will not cover the code blocks for these, refer to later sections (see headings tab) for the implementations.

The adaptations which will be implemented are asexual reproduction, predation, and terrain specific benefits (such as bonus speed).

3.5.2.1 Asexual reproduction

An organism will gain the ability to asexually reproduce after a minimum of 3 generations of female organisms which have had less than 10 interactions with other fecund organisms throughout their life spans.

For this to happen, the organisms will have to store their generation, starting at 1, and going for as long as the user runs the simulation. The organisms will also have to store their total organisms encountered, and their parents IDs too. Therefore, each time organisms breed, this information is checked against. If the requirements are met (see above), then the organism will adapt the ability to asexually reproduce, becoming independent from all other organisms. This will of course affect their survivability in different terrains, and they will be more inclined to not go towards other organisms. If there is an organism which goes through asexual evolution, they will be able to reproduce much faster, potentially leading to predators seeking them out over other organisms.

3.5.2.2 Predation

An organism will gain the ability to become a predator (seeks and attacks other organisms). This will come as a result of 2 generations of organisms which interact with over 50 organisms (the opposite of the requirements needed for asexual reproduction). Organisms which eat other animals by default (such as deep-sea fish) will be able to attack other organisms by default, however, will not actively seek them out unless coming into contact with them often. An organism which eats other animals will be able to attack and eat other organisms during the first generation if it meets more than 10 other organisms (rare).

The implementation of this section will be covered in 3.6 as it is a separate feature to organism breeding.

3.5.2.3 Terrain Specific Benefits

An organism which is the fifth-generation organism to have the same preferred habitat as its line of fathers will have an increased speed and awareness within its environment, this means that these organisms are much more likely to survive and reproduce through finding other organisms, especially if they are predators.

Once again this will require the storage of an organism's parent IDs, like 3.5.1.1. The organisms will have a speed, health, and an awareness increase. This will also mean that these organisms are highly likely to become predators in the future because of the increased speed and awareness.

3.5.3 Implementing evolutionary adaptations

Each section here gives the logic behind evolutionary adaptations, as well as the code and the tests. Refer to the headings section to skip to specific sections.

3.5.3.1 Asexual reproduction

This will cover the logic which was designed in 3.5.2. The organism will be checked for its parents' total encounters, as well as which generation it is. If it matches the criteria laid out within 3.5.2, then the organism will be able to asexually reproduce, with its sex now being present as asexual. This means that the organism may give birth to a child organism every 5 years after the age of 15. These organisms

however will be much less inclined to evolution, as they only have 1 parent, the gene pool is significantly decreased and therefore evolution is slower. The asexually produced organisms may die faster over time, however there will be a much larger number of them.

See the code block below for the method updating this.

```

1.      // Checks if the organism should become asexual
2.      // Checks the mothers of the organism for the number of fecund organisms encountered, if it
is less than 10 each, the organism can asexually reproduce
3.      public void checkAsexual(){
4.          int generationsToCheck = 5; // The number of generations of mothers to check
5.          Organism currentOrganism = organisms[this.organismID]; // The organism being operated
on
6.
7.          // If the organism is water based then there is a significant boost in the chance it
becomes asexual
8.          if(Objects.equals(currentOrganism.getCurrentTerrain(), "Deep Water") ||
Objects.equals(currentOrganism.getCurrentTerrain(), "Water")) generationsToCheck = 3;
9.
10.         // Check the generations of mothers for how many organisms have been encountered
11.         if(stats.getGeneration() >= generationsToCheck) {
12.             int totalEncounters = 0;
13.             List<Organism> organismsToCheck = new ArrayList<>(); // List of mothers to check
14.             for (int counter = 0; counter <= generationsToCheck; counter++) {
15.                 organismsToCheck.add(organisms[currentOrganism.getMotherID()]); // Add the
immediate mother
16.                 currentOrganism = organisms[currentOrganism.getMotherID()]; // Set the current
organism to be the mother
17.             }
18.             for(int secondCounter = 0; secondCounter <= organismsToCheck.size();
secondCounter++){ // Add up the total encounters
19.                 totalEncounters = totalEncounters +
organismsToCheck.get(secondCounter).getStats().getOrganismsMet();
20.             }
21.             if(totalEncounters <= 10 * generationsToCheck){ // If the number of encounters are
less than the 10 each
22.                 organisms[this.organismID].setSex(2); // Set to be asexual
23.             }
24.         }
25.     }

```

Java

3.5.3.2 Testing asexual reproduction

This may be difficult to test, as it is not guaranteed asexual reproduction to become present. However, I will run through simulations until an organism becomes asexual, once it does, it can be analysed to determine the success of this section. If the organism's parents have each met a maximum of 10 organisms, and the organism itself is a third-generation organism, then this section is a success. See below for the test results.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.8.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=53499:C:\Program Files\JetBrains\
\IntelliJ IDEA Community Edition 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\
Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.
jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\
IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\
lib\jackson-core-2.13.3.jar Main
2 66 has become asexual
3 70 has become asexual
4 59 has become asexual
5 72 has become asexual
6 75 has become asexual
7 77 has become asexual

```

As shown here, the simulation has proven that organisms have the ability to become asexual, quite frequently as well. For this one, we will examine organism 66 (line 2). Below is the JSON which will be checked.

```
1. {
2.     "organismId": 66,
3.     "stats": {
4.         "currentAwareness": 1.7,
5.         "organismsEncountered": 0.5,
6.         "awarenessHistory": [*],
7.         "age": 70,
8.         "preferredFood": {"foodName": "Beach Grass Seeds", *},
9.         "preferredFoodString": null,
10.        "eatenFood": {*} ,
11.        "recentFood": [],
12.        "foodFactor": 0.15,
13.        "sizeFactor": 34.5,
14.        "size": 0.6,
15.        "speed": 12.89,
16.        "startSpeed": 5.5,
17.        "currentTemp": 8,
18.        "preferredTemp": 7,
19.        "recentTemp": [],
20.        "tempFactor": 0.96,
21.        "preferredTerrain": "Water",
22.        "secondPreferredTerrain": "Beach",
23.        "thirdPreferredTerrain": "Default",
24.        "health": 0.0,
25.        "dead": true,
26.        "ageLastBred": -1,
27.        "generation": 3,
28.        "organismsMet": 1
29.    }
30.    "currentTerrain": "Water",
31.    "xCoordinate": 859,
32.    "yCoordinate": 371,
33.    "sex": 1,
34.    "fatherID": 53,
35.    "motherID": 1
36. }
```

* Shows omitted data

JSON

From this, we can evaluate that the organism is indeed the third generation of organisms. The highlighted factors are the factors that are going to be tested here.

The generation, as mentioned, is the third generation, with the preferred terrain being water, meaning so long as the previous female ancestors also have a preferred terrain of water the criteria is met here. The organisms met for this organism is 1, while not particularly of note for this organism, it will become important when looking at the father and mother. This is why the mother's ID has also been highlighted in the JSON.

The mother organism (ID 1), while being generation 1, lived for 71 years and met 3 other organisms. Due to the mother being the first generation, it has no mother to check. This has prompted me to change the criteria for organisms to become asexual. For an organism to become asexual now, it must be a generation of 8 or higher. This ensures that there is less chance for a boosted evolution, instead, there is a correct chain of organisms which are checked, reducing the number of organisms which will become asexual while keeping a robust criteria. Despite organisms gaining this evolutionary adaptation more frequently than expected, once organisms seek out each other (see 3.6 predator prey cycles), this will become much harder to achieve as organisms will have many more encounters with organisms they can breed with.

Evidently, this evolutionary adaptation works as expected, with organisms that match the criteria becoming asexual.

While testing the asexual organism's ability to breed, I noticed that if there are more than 1 organism created in a year, all organisms created have the same ID, for example in the JSON for the above test, there were 3 organisms with an ID of 66. This will pose a threat for finding specific organisms, as the ID is not robust. To change this however was very simple. A global variable keeping track of the size of the JSON was introduced, with it being incremented once a new organism is created so the next organism has the correct ID. Upon further testing the organisms having the same ID was no longer a problem and therefore no longer an error requiring attention.

3.5.3.3 Terrain specific benefits

This will cover the logic laid out in 3.5.1.3. If the organism is a fifth generation (or above) organism, then it will compare its parents and its own preferred terrain. If they are all similar, then the organism will receive the evolutionary bonus for that terrain. During testing (3.5.3.4), the bonuses may change if they are too much of an overkill in the balance of organisms. See below for the method checking this.

```

1.    // Checks if the organism should have terrain specific benefits
2.    // Checks 5 generations of fathers, if they all have the same preferred terrain,
3.    // then the organism will have terrain specific benefits (TSB) for that terrain
4.    public void checkTSB(){
5.        int generationsToCheck = 5; // Number of generations to check
6.        Organism currentOrganism = organisms[this.organismID]; // The organism to be operated
on
7.        String preferredTerrain = currentOrganism.getStats().getPreferredTerrain(); // The
current organisms preferred terrain
8.        int terrainLog = 0; // Tracks how many of the fathers have the same preferred terrain
9.
10.       if(stats.getGeneration() >= generationsToCheck) { // If the generation is old enough to
have TSB
11.           List<Organism> organismsToCheck = new ArrayList<>(); // The list of father
organisms to check
12.           for(int counter = 0; counter <= generationsToCheck; counter++){ // Populates the
list
13.               organismsToCheck.add(organisms[currentOrganism.getFatherID()]); // Adds the
father to the list
14.               currentOrganism = organisms[currentOrganism.getFatherID()]; // Sets the current
one to operate on to the father of the previous
15.           }
16.           for(int counter2 = 0; counter2 <= organismsToCheck.size(); counter2++){ // Checks
the preferred terrain of each
17.
if(preferredTerrain.equals(organismsToCheck.get(counter2).getStats().getPreferredTerrain())) {
18.                terrainLog++; // Increments if the preferred terrain is the same
19.
}
20.
}
21.           if(terrainLog == organismsToCheck.size()) stats.setTSB(preferredTerrain); // All
preferred terrains are the same and therefore organism will have TSB for that terrain
22.
}
23.    }

```

Java

For the organisms to make use of the TSB, the food factor if they are in the terrain will increase by 40% of the default, health will increase by 0.2, and the current awareness will increase also by 0.2. See the statements below which implement these in the appropriate locations.

```

1.    private void updateHealth() {
2.        // . . . Other code
3.
4.        if(Objects.equals(stats.getTSB(), organisms[organismID].getCurrentTerrain())) {
5.            totalDecrease = totalDecrease + 0.2; // Decrease the health by less
6.
}
7.        // . . . Other code

```

```

8.    }
9.
10.   private void updateFood(TerrainAttributes terrainAttributes) {
11.     // . . . Other code
12.
13.     if(Objects.equals(organisms[organismID].getCurrentTerrain(), stats.getTSB())) {
14.         foodFactor = foodFactor + (foodFactor * 0.4); // Add additional food factor
15.     }
16.
17.     // . . . Other code
18.   }
19.
20.   private void updateAwareness(TerrainAttributes terrainAttributes) {
21.     // . . . Other code
22.
23.     if(Objects.equals(stats.getTSB(), organisms[organismID].getCurrentTerrain())) {
24.         currentAwareness = currentAwareness + 0.2; // Raise the current awareness
25.     }
26.     // . . . Other code
27.   }

```

Java

3.5.3.4 Testing terrain specific benefits

For this, once the criteria for terrain specific benefits are met, the program will display the preferred terrain of the organism's parents. If all are the same, and the organism in question is a fifth-generation organism, then this section is part success. If the traits that are given as a bonus to the organism (as a result of terrain specific benefits) are too powerful (organism can move too far) then the amount increased by will need to be changed. If not, then the terrain specific benefits are implemented.

```

1. {
2.   "organismId": 77,
3.   "stats": {
4.     "currentAwareness": 1.6,
5.     "organismsEncountered": 0.5,
6.     "awarenessHistory": [],
7.     "age": 71,
8.     "preferredFood": {
9.       "foodName": "Grass Seeds",
10.      "foodAvailability": 0.6,
11.      "foodFactor": 0.3,
12.      "waterBasedFood": false,
13.      "landBasedFood": true,
14.      "parentName": "Plant Spawn",
15.      "terrainName": "Grass"
16.    },
17.    "preferredFoodString": null,
18.    "eatenFood": {
19.      "foodName": "Insects and Insect Larvae",
20.      "foodAvailability": 0.2,
21.      "foodFactor": 0.5,
22.      "waterBasedFood": true,
23.      "landBasedFood": false,
24.      "parentName": "Aquatic life / Spawn",
25.      "terrainName": "Water"
26.    },
27.    "recentFood": [],
28.    "foodFactor": 0.15,
29.    "sizeFactor": 35.25,
30.    "size": 0.7,
31.    "speed": 9.51,
32.    "startSpeed": 4.0,
33.    "currentTemp": 8,
34.    "preferredTemp": 9,
35.    "recentTemp": [],
36.    "tempFactor": 0.96,
37.    "preferredTerrain": "Grass",
38.    "secondPreferredTerrain": "Water",
39.    "thirdPreferredTerrain": "Beach",
40.    "health": 0.0,

```

```

41.      "dead": true,
42.      "ageLastBred": 38,
43.      "generation": 5,
44.      "organismsMet": 17,
45.      "tsb": "Grass"
46.    },
47.    "currentTerrain": "Water",
48.    "xCoordinate": 600,
49.    "yCoordinate": 511,
50.    "sex": 2,
51.    "fatherID": 59,
52.    "motherID": 74
53.  }
54.

```

The organism in the JSON here has a terrain specific benefit (line 45 “tsb”) for Grass. It is shown in the JSON that the organism is a fifth generation organism (before the implementation of the 8 generation rule in 3.5.2), meaning the criteria is met there. Regarding the line of fathers, within the JSON, checking the ID of 59 and 12 (before the implementation of 8 generation rule ensuring at least 3 checkable parents), both fathers also had a preferred terrain of Grass, meaning the criteria for this evolutionary adaptation is correct.

If the increases to health, food factor, and awareness prove to be too advantageous for organisms once predator prey cycles are implemented, then these values may change.

3.6 Adapted organism movement

As mentioned within 3.5, the movement of organisms is not only diagonally biased, but only takes into account the terrain surrounding the organism. This will have to change not only for organisms to breed as intended, but also for predator organisms to chase, and for prey organisms to run. This section covers the adaptation of 3.4 Organism Movement to allow these factors into the program.

3.6.1 Correcting the diagonal bias

Unfortunately, during the testing of 3.4, the testing of theory was not extensive enough and did not cover movement physically. While the survivability was checked, the actual positions which organisms could move to, was not. This means that when the list of potential x and y points were created, it added both at the same time and therefore an organism could only move diagonally. To correct this, I will iterate through one of the coordinates, adding in each x and y for each, creating a full list. The original code is shown below.

```

1.      // The calculated decision made by each organism to move to the best position based on
survivability
2.      private void makeDecision() {
3.          // Lists which will contain the x and y coordinates which can be moved to
4.          List<Point> positions = new ArrayList<>();
5.
6.          // Populates the list created above with the appropriate coordinates
7.          for (int xCounter = (int) (0 - organismSpeed); xCounter < organismSpeed; xCounter++) {
8.              int newX = Math.abs(currentX + xCounter);
9.              int newY = Math.abs(currentY + yCounter);
10.             positions.add(new Point(newX, newY));
11.         }
12.         // . . . Other code
13.     }

```

Java

As shown in this, both the newX and newY were added to the list at the same time, instead of each y coordinate within x, or visa versa. The correction for this is shown below.

```

1.      // The calculated decision made by each organism to move to the best position based on
survivability
2.      private void makeDecision() {
3.          // Lists which will contain the x and y coordinates which can be moved to

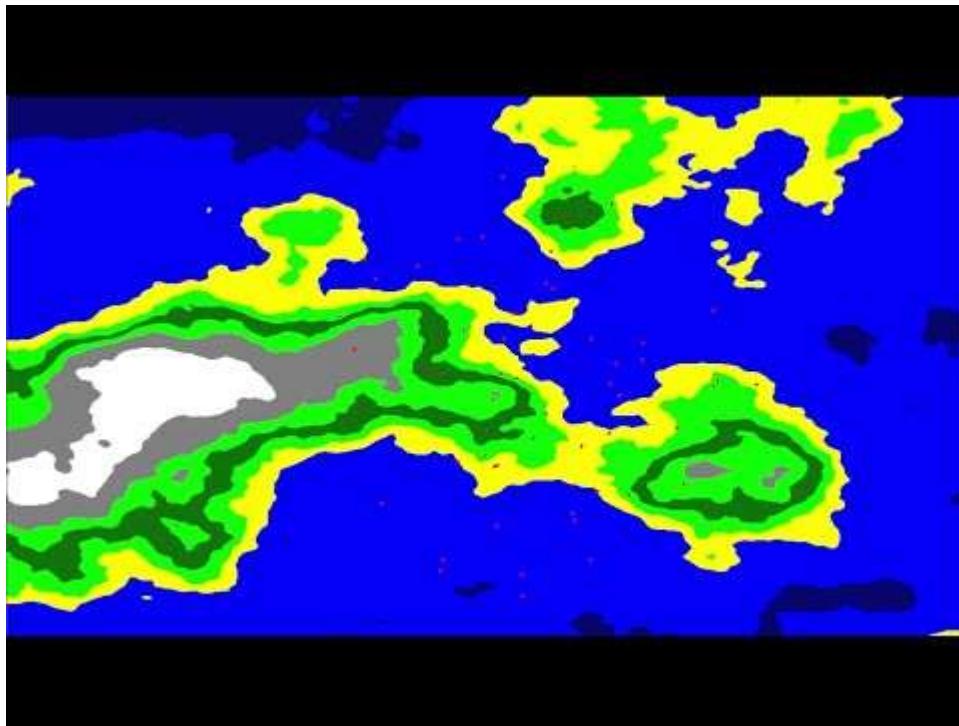
```

```

4.     List<Point> positions = new ArrayList<>();
5.
6.     // Populates the list created above with the appropriate coordinates
7.     for (int xCounter = (int) (0 - organismSpeed); xCounter < organismSpeed; xCounter++) {
8.         int newX = Math.abs(currentX + xCounter);
9.         for (int yCounter = (int) (0 - organismSpeed); yCounter < organismSpeed;
yCounter++) { // Iterates through y coordinates along point x
10.             int newY = Math.abs(currentY + yCounter);
11.             positions.add(new Point(newX, newY));
12.         }
13.     }
14.     // . . . Other code
15. }
```

Java

Shown in the video below, are the organisms moving around freely as a result of the corrected code.



3.6.2 Organism movement based on other organisms

This section will not only benefit the predator/prey cycle, but also the breeding process. This means that during breeding, organisms that are old enough will attempt to move closer to organisms which they can breed with, and further away from ones which they cannot. For the predator prey cycles, the prey will attempt to get as far away as possible from the predators, with the predators chasing. This can be achieved through changing the survivability of certain coordinates which are close to target organisms, for example if a coordinate is the same as another organism's which the current organism can breed with, the survivability gets a bonus and therefore has more of a chance for the organism to move to it.

For this, I will get the Euclidean distance, through the formula which is shown below. This will return the distance between the potential move, and the organism which it is seeking, the smaller the distance, the higher the survivability.

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The implementation of listing the closest organisms is shown below. The function takes in the current organism, creates a list, and fills it with other organisms ordered by the distance. Please refer to the comments within as these explain what the sections do

```

1.    // This returns a list of closest organisms to organism sent in
2.    // Returns a list of points which contains an x and y coordinate
3.    private List<Organism> listClosestOrganisms(Organism currentOrganism) {
4.        List<Organism> closestOrganisms = new ArrayList<>();
5.        Point currentLocation = new Point(currentOrganism.getxCoordinate(),
currentOrganism.getyCoordinate());
6.
7.        // Filter and sort Organisms to Points which are closest to the current organism
8.        // Uses .stream to process the array
9.        // Filters out organisms that have the same ID and are dead
10.       // Maps each organism to a Point using the organisms x and y coordinates
11.       // Sorts the points created and converts into a list
12.       List<Point> organismPoints = Arrays.stream(organisms)
13.           .filter(organism -> organism.getOrganismId() != currentOrganism.getOrganismId()
&& !organism.getStats().isDead())
14.           .map(organism -> new Point(organism.getxCoordinate(),
organism.getyCoordinate())).sorted(Comparator.comparingDouble(p ->
Math.sqrt(Math.pow(currentLocation.x - p.x, 2) + Math.pow(currentLocation.y - p.y, 2)))).toList();
15.
16.       // Transfers the list of points into a list of correlating organisms
17.       for (Point point : organismPoints) {
18.           for (Organism organism : organisms) {
19.               if (organism.getxCoordinate() == point.x && organism.getyCoordinate() == point.y && !organism.getStats().isDead()) {
20.                   closestOrganisms.add(organism);
21.                   break;
22.               }
23.           }
24.       }
25.
26.       return closestOrganisms; // Returns the list of closest organisms
27.   }

```

Java

This function is called when the program wants to find an organism to seek. The function `organismToSeek` returns the target organism, based on the current organisms attributes such as whether it can breed, or if it is a predator or not. The code block below shows this function, alongside relevant comments explaining the section.

```

1.    // This will cover the organisms selection of organism to seek out, whether it be as prey
or as breeding
2.    public Organism organismToSeek(){
3.        Organism currentOrganism = organisms[this.organismID]; // Localises the current
organism
4.        List<Organism> closestOrganisms = listClosestOrganisms(currentOrganism); // Creates the
list of closest organisms
5.        Organism targetOrganism = new Organism(); // Initialising the current organism
6.
7.        if(currentOrganism.isPredator() && !(stats.getAge() >= 15) && !(stats.getAgeLastBred() <
stats.getAge() - 5)){ // If the organism is a predator and cannot breed
8.            for(Organism organism : closestOrganisms){
9.                if(!organism.isPredator()){
10.                    targetOrganism = organism; // Target the closest organism as prey
11.                    break;
12.                }
13.            }
14.        } else if(!currentOrganism.isPredator() && !(stats.getAge() >= 15) &&
! (stats.getAgeLastBred() < stats.getAge() - 5)){ // If the organism is not a predator and cannot
breed
15.            targetOrganism = null; // Do not target for another organism
16.        } else if(!currentOrganism.isPredator() && stats.getAge() >= 15 &&
stats.getAgeLastBred() < stats.getAge() - 5){ // If the organism is not a predator and can breed
17.            for(Organism organism : closestOrganisms){
18.                if(!organism.isPredator() && organism.getStats().getAge() >= 15 &&
organism.getStats().getAgeLastBred() < organism.getStats().getAge() - 5 && organism.getSex() != 2
&& organism.getSex() != currentOrganism.getSex()){ // If the other organism is not a predator, not
the same sex, and not asexual
19.                    targetOrganism = organism; // Find the closest organism it can breed with
20.                    break;
21.                }

```

158

```

22.         }
23.     } else {
24.         targetOrganism = closestOrganisms.get(0); // Target the closest organism to breed
with or attack
25.     }
26.
27.     return targetOrganism; // Return the target organism
28.   }

```

Java

The organism to seek is found when the survivability of an area is needed. As mentioned earlier, the survivability of one section is demonstrated by how close it is to the target organism. The following code block not only counts for terrain, but also the organism which is being sought out. Refer to comments within the code block below for specific explanations.

```

1.    // Gets the survivability of an organism at the coordinate position newX, newY
2.    public double getSurvivability(int newX, int newY, Organism organismToSeek) {
3.        double survivability = 0.1; // Default value
4.        double chanceToMisjudge = random.nextDouble(0, 1); // Small chance to make a mistake
5.        String newTerrain = generateMap.getTerrain(newX, newY); // Gets the terrain at position
newX, newY
6.
7.        if(chanceToMisjudge <= 0.02) { // Small chance to misjudge
8.            if (organismToSeek != null) { // If it is not null (no other organisms)
9.                // Calculate the Euclidean distance
10.               double currentDistance = Math.sqrt(Math.pow(organismToSeek.getxCoordinate()
- newX, 2) + Math.pow(organismToSeek.getyCoordinate() - newY, 2));
11.
12.               survivability = (1.0 / (1.0 + currentDistance)) * 10; // Make it between -2
and 2
13.               survivability =
Double.parseDouble(decimalFormat.format(Math.max(Math.min(survivability - 1, 1.5), 0))); // Limit
it to 0 and 1.5
14.           }
15.
16.           if (newTerrain.equals(organisms[this.organismID].getStats().getPreferredTerrain()))
17.               survivability = survivability + 0.15; // If it is the preferred terrain receive
a boost
18.           else if
(newTerrain.equals(organisms[this.organismID].getStats().getSecondPreferredTerrain()))
19.               survivability = survivability + 0.10; // If it is the second most preferred
terrain receive a boost
20.           else if
(newTerrain.equals(organisms[this.organismID].getStats().getThirdPreferredTerrain()))
21.               survivability = survivability + 0.05; // If it is the third most preferred
terrain receive a boost
22.           } else survivability = 1.5; // Default to 1.5 as a misjudgement
23.
24.       return survivability; // Returns the calculated survivability
25.   }

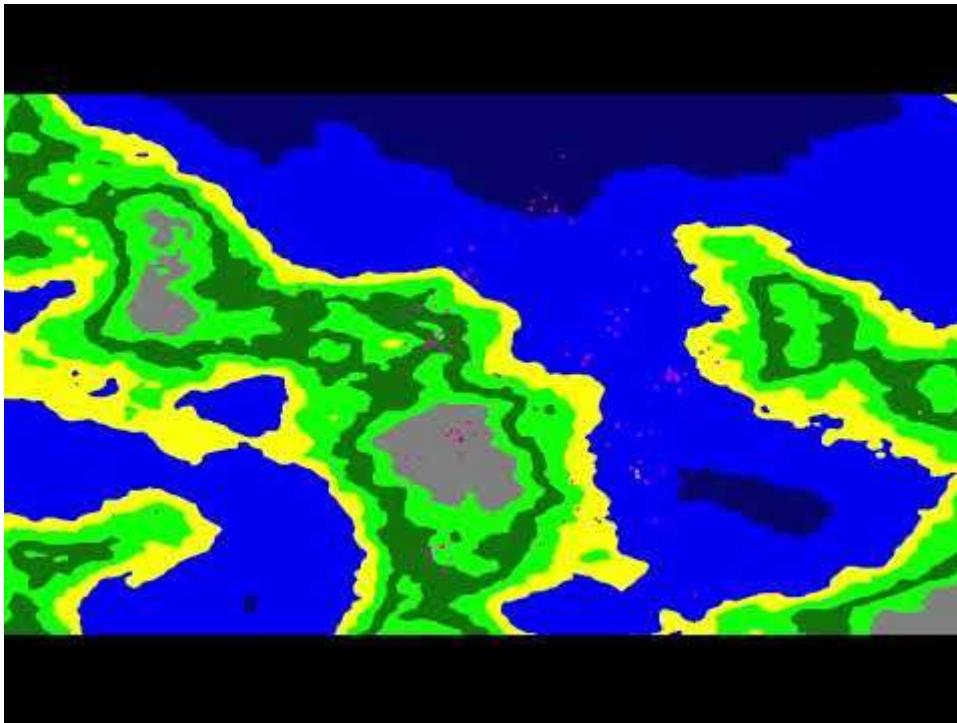
```

Java

The previous method for getting the final coordinate to move to is unchanged, the list of survivability is ordered, highest syphered and a random one selected and set.

3.6.3 Testing the adapted organism movement

In the video below, it is clear that organisms frequently breed, with them seeking out an organism and breeding accordingly. Due to the organisms seeking out other organisms it means it is now theoretically possible to run a full simulation with 2 organisms, however it is very difficult to have generations grow off of 2 organisms, especially if there are children of the same gender.



At the start of the video above, it can be seen that many of the organisms come close / direct contact with other organisms, and then proceed to breed. This shows that the adapted movement is working as intended, with the organisms moving closer when they want to breed, and moving randomly if they can't breed. It is also clear however later into the video that they are breeding too frequently, potentially requiring limits being imposed for breeding. Predator / prey cycles however should significantly cut down on the amount of organisms breeding, as they attack and weaken other organisms faster.

3.7 Predator / Prey Cycles

As mentioned in 3.5.1.2, the organisms can adapt and become a predator, if 2 previous generations of organisms have met more than 20 organisms during their life each.

3.7.1 Logic

Once an organism becomes a predator, it will have the ability to seek out other organisms for the purpose of attempting an attack (see 3.6 for the movement logic). Once the predator is within range of the 'prey' organism, it will attack the organism (attack and defence attribute required), decreasing the health of both organisms, but increasing to whomever wins. For example, if an organism with an attack attribute of 2 (2 previous attacks), attacks an organism with a defence of 3 (previously attacked 3 times), the predator organism will lose, with the health bonus going to the prey organism, and the predator taking a health hit. Once the predator organism has attacked another organism, it cannot attack for another 5 years (to avoid repeat attacks).

3.7.2 Allowing organisms to become predators

The code block below checks if the organism should become a predator, using the logic from 3.7. If 2 generations of father organisms have met 20 or more other organisms, then the current organism is a predator as well. Refer to the comments within the code block for explanations.

```
1. // This method checks if the organism should be a predator if it isn't already
2. private void checkPredation(){
3.     int generationsToCheck = 5; // Number of generations to check
4.     Organism currentOrganism = organisms[this.organismID]; // The organism to be operated
on
```

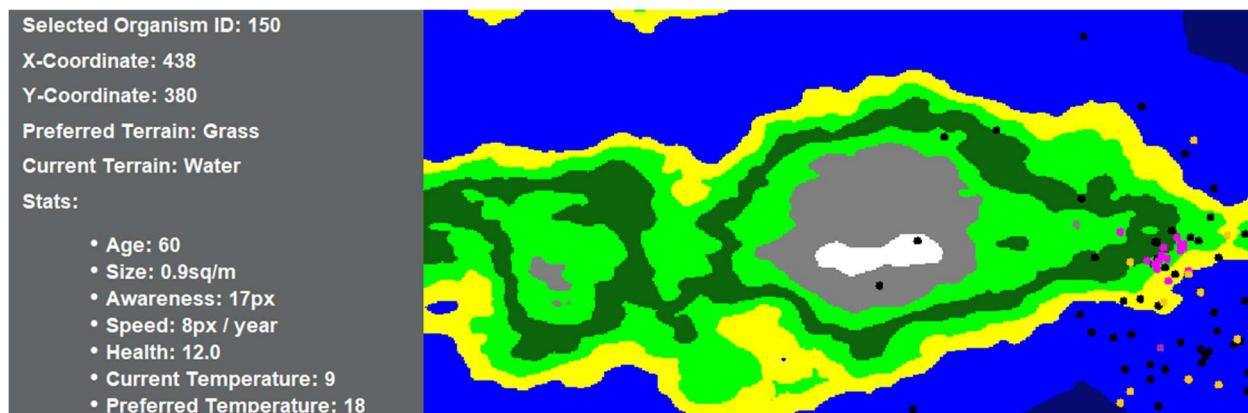
```

5.
6.         if(stats.getGeneration() >= generationsToCheck) { // If the organism is old enough
7.             int totalEncounters = 0; // To count the number of organisms met
8.             generationsToCheck = generationsToCheck - 3;
9.             List<Organism> organismsToCheck = new ArrayList<>(); // The list of father
organisms to check
10.            for (int counter = 0; counter < generationsToCheck; counter++) { // Populates the
list
11.                if (currentOrganism.getFatherID() != -1) {
12.                    organismsToCheck.add(organisms[currentOrganism.getFatherID()]); // Adds the
father to the list
13.                    currentOrganism = organisms[currentOrganism.getFatherID()]; // Sets the
current one to operate on to the father of the previous
14.                }
15.            }
16.            for (Organism organism : organismsToCheck) { // Add up the total encounters
17.                totalEncounters = totalEncounters + organism.getStats().getTotalOrganismsMet();
18.            }
19.            if (totalEncounters >= 20 * generationsToCheck) { // If the number of encounters
are less than the 20 each
20.                currentOrganism.setPredator(true); // Organism becomes a predator
21.            }
22.        }
23.    }
24.

```

3.7.3 Testing if the organisms can become a predator

In the image below, the dark purple organism has been coloured that way due to it being marked as a predator. As seen from the side menu, the selected organism ID is 150.



Referring this to the JSON, the two previous father organisms (ID 130 and ID 114), both have had well over 20 interactions, seen from the total organisms met factor. This means that organisms are likely being made a predator incredibly common. To fix this, I will simply change the requirements as needed to change the chance that organisms become a predator. This will take place towards the end of development alongside other balancing.

```

{
  "motherID": 114, "organismId": 130,
  "dead": true, "preferredFood": {"waterBasedFood": true, "landBasedFood": false}, "ageLastBred": 66, "recentFood": [{"id": 1, "x": 1.1, "y": 1.1}], "foodFactor": 0.2, "terrainName": "Water", "foodAvailability": 0.4, "totalOrganismsMet": 176, "health": 12.0, "secondPreferredTerrain": "Grass", "currentAwareness": 1.6}, "predator": true
}
{
  "xCoordinate": 438, "yCoordinate": 380, "organismId": 150, "motherID": 135, "dead": false, "preferredFood": {"waterBasedFood": true, "landBasedFood": false}, "ageLastBred": 66, "recentFood": [{"id": 1, "x": 1.1, "y": 1.1}], "foodFactor": 0.25, "terrainName": "Water", "foodAvailability": 0.4, "totalOrganismsMet": 416, "health": 12.0, "secondPreferredTerrain": "Grass", "currentAwareness": 1.6}, "predator": true
}

```

3.7.4 Organisms attacking

```
1.    // The decision between which organism wins during an attack
2.    private void newOrganismAttack(Organism attackingOrganism, Organism defendingOrganism) {
3.        double attackLevel = attackingOrganism.getStats().getAttack(), defenceLevel =
defendingOrganism.getStats().getDefence(); // The attack and defence points
4.        int attackerHealth = -5, defenderHealth = -5; // Default health decrease
5.
6.        if (organisms[attackingOrganism.getOrganismId()].getStats().getLastAttack() <
organisms[attackingOrganism.getOrganismId()].getStats().getAge() - 10) {
7.            // Increase their abilities equally
8.            if (attackLevel > defenceLevel) { // If the attacker wins
9.                System.out.println(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and " + attackingOrganism.getOrganismId() + " has won"); //
To test
10.               defenderHealth = defenderHealth - random.nextInt(7, 13); // Defender health
high decrease
11.               attackerHealth = attackerHealth - random.nextInt(3, 7); // Attacker low health
decrease
12.               organisms[attackingOrganism.getOrganismId()].getStats().setFoodFactor(1); //
Give the attacking organism other benefits for that year
13.               organisms[attackingOrganism.getOrganismId()].getStats().setTempFactor(1);
14.
organisms[attackingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[attackingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2);
15.           } else {
16.               System.out.println(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and " + defendingOrganism.getOrganismId() + " has won"); //
To test
17.               attackerHealth = attackerHealth - random.nextInt(7, 13); // Attacker health
high decrease
18.               defenderHealth = defenderHealth - random.nextInt(3, 7); // Defender low health
decrease
19.               // Give the organism an awareness increase
20.
organisms[defendingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[defendingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2);
21.           }
22.           // Increase their abilities and update the health
23.
organisms[attackingOrganism.getOrganismId()].getStats().setAttack(organisms[attackingOrganism.getOrganismId()].getStats().getAttack() + 0.5);
24.
organisms[attackingOrganism.getOrganismId()].getStats().setHealth(organisms[attackingOrganism.getOrganismId()].getStats().getHealth() + attackerHealth);
25.
organisms[defendingOrganism.getOrganismId()].getStats().setDefence(organisms[defendingOrganism.getOrganismId()].getStats().getDefence() + 1);
26.
organisms[defendingOrganism.getOrganismId()].getStats().setHealth(organisms[defendingOrganism.getOrganismId()].getStats().getHealth() + defenderHealth);
27.
organisms[attackingOrganism.getOrganismId()].getStats().setLastAttack(organisms[attackingOrganism.getOrganismId()].getStats().getAge()); // Set the attacking organisms age to its current age
28.       }
29.   }
30.
```

The code block above demonstrates how an attack would play out. It follows with the logic defined in 3.7.1, where if the attack attribute is higher than the others defence attribute, then the attacking organism wins, and visa versa. Refer to the comments within for explanation on the specific sections. Note that the attacking organism only gains 0.5 attack (instead of 1) in attempt to balance the result.

The method above is called in a similar way to breeding, where if 2 organisms meet within close proximity they are checked if they can breed, and if not they are checked if they can attack. If both organisms are predators, then a random one initiates the attack. Currently the winning organism is only ranked based off of how many attacks it has been through previously, however this may change after testing if there is no balance between winners.

3.7.5 Testing the attacks

Using the test statements added in above the console log can be analysed to test the attacks. The organism which initiated the attack can be checked, as can the defendant, with both being compared to see who should win.

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.9\bin\java.exe "-javaagent:C:\Program  
Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.5\lib\idea_rt.  
jar=51119:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition  
2023.2.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\  
IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-  
simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-  
databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-  
annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-  
core-2.13.3.jar Main  
2 121 has attacked 245 and 245 has won  
3 154 has attacked 372 and 372 has won  
4 155 has attacked 143 and 143 has won  
5 151 has attacked 341 and 341 has won  
6 147 has attacked 162 and 162 has won  
7 187 has attacked 353 and 353 has won  
8 199 has attacked 143 and 143 has won  
9 155 has attacked 173 and 155 has won
```

From this, it is clear there is a disproportion between the attacks, with the defending organism almost always winning. This has prompted me to stray away from the simple system of checking attack and defence attributes, instead making use of the organisms size and speed. If the size and of speed of the attacker are greater, then the attacking organism will win. If both are less than the defending organism, then the defendant wins. If one has a larger size but the other has a larger speed, then it is a draw. To avoid this, the age of both organisms will be compared, the younger one receiving a slightly smaller bonus.

3.7.6 Revising the attacking method

The difference between the organisms size, speed, and age will be compared. This means there is a greater chance for the more developed organisms to win a fight, with the fighting logic also being much more robust. See below for the redesigned code block.

```
1. // The decision between which organism wins during an attack  
2. private void newOrganismAttack(Organism attackingOrganism, Organism defendingOrganism) {  
3.     double attackerSpeed, defenderSpeed, attackerSize, defenderSize; // Defining the  
attributes to be used  
4.     int attackerAge, defenderAge;  
5.     int attackerHealth = -5, defenderHealth = -5; // Default health decrease  
6.  
7.     if (organisms[attackingOrganism.getOrganismId()].getStats().getLastAttack() <  
organisms[attackingOrganism.getOrganismId()].getStats().getAge() - 10) {  
8.         // Initialising the attributes  
9.         attackerSpeed = organisms[attackingOrganism.getOrganismId()].getStats().getSpeed();  
10.        attackerSize = organisms[attackingOrganism.getOrganismId()].getStats().getSize();  
11.        attackerAge = organisms[attackingOrganism.getOrganismId()].getStats().getAge();  
12.        defenderSpeed = organisms[defendingOrganism.getOrganismId()].getStats().getSpeed();  
13.        defenderSize = organisms[defendingOrganism.getOrganismId()].getStats().getSize();  
14.        defenderAge = organisms[defendingOrganism.getOrganismId()].getStats().getAge();  
15.  
16.        // Getting the total difference between the two. If the number is negative, the  
defendant has better attributes  
17.        double totalScore = 0;  
18.        totalScore = totalScore + (attackerSpeed - defenderSpeed);  
19.        totalScore = totalScore + (attackerSize - defenderSize);  
20.        totalScore = totalScore + (attackerAge - defenderAge);  
21.  
22.        if (totalScore >= 0) { // The attacker has won, give the attacker benefits  
23.            attackerHealth = attackerHealth - random.nextInt(7, 13); // Attacker health  
high decrease  
24.            defenderHealth = defenderHealth - random.nextInt(3, 7); // Defender low health  
decrease
```

```

25.           // Give the attacker an awareness increase
26.
27.           organisms[attackingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[attackingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2);
28.       } else { // The attacker has lost, give the defender benefits
29.           attackerHealth = attackerHealth - random.nextInt(7, 13); // Attacker health
30.           defenderHealth = defenderHealth - random.nextInt(3, 7); // Defender low health
31.           decrease
32.           organisms[defendingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[defendingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2); // Give the defender an awareness
33.           increase
34.       }
35.
36.       // Increase their abilities and update the health
37.
38.       organisms[attackingOrganism.getOrganismId()].getStats().setHealth(organisms[attackingOrganism.getOrganismId()].getStats().getHealth() + attackerHealth);
39.   }

```

Java

This code block, while slightly bigger, provides a much better method for calculating the organisms attack, as it takes into account the organism attributes themselves, making use of previous algorithms and stats. The difference between the attributes such as size is a key indicator of which organism should have a better chance at survival. In real life, smaller organisms have much less chance against larger organisms, however faster organisms have a better chance against slower/larger organisms. By calculating the differences, the simulation is more of a reflection to real life (making it a simulation), which is what the aim is for EvSim.

3.7.8 Testing the new attacks

Once again, I will use console logs to check the methods validity. If the winning organism has better attributes compared to the other, then the method is correct and works as intended.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.9\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2023.2.5\lib\idea_rt.jar=51428:C:\Program Files\JetBrains\IntelliJ IDEA Community
Edition 2023.2.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\
production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\
jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\\
Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 -21.009999999999998
3 127 has attacked 96 and 96 has won
4 -24.58
5 176 has attacked 137 and 137 has won

```

The value that is sent is the resultant total score. It is clear from this that the program has gone the other way. This is due to the size and speed being measured differently. To counter this, I will change to have 3 scores. This will require a few more IF/ELSE statements however will correct the error found above.

```

1. // Getting the differences between the attributes
2.     double speedScore = (attackerSpeed - defenderSpeed);
3.     double sizeScore = (attackerSize - defenderSize);
4.     int ageScore = (attackerAge - defenderAge);
5.
6.     if ((speedScore >= 0 && sizeScore >= 0) || (sizeScore >= 0 && ageScore < 20) ||
(speedScore >= 0 && ageScore < 20)) { // The attacker has won, give the attacker benefits

```

```

7.         attackerHealth = attackerHealth - random.nextInt(3, 7); // Attacker health high
decrease
8.         defenderHealth = defenderHealth - random.nextInt(7, 13); // Defender low health
decrease
9.         // Give the attacker an awareness increase
10.
organisms[attackingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[attackingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2);
11.         System.out.println(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and " + attackingOrganism.getOrganismId() + " has won");
12.     } else { // The defendant has won, give the defendant benefits
13.         attackerHealth = attackerHealth - random.nextInt(7, 13); // Attacker health
high decrease
14.         defenderHealth = defenderHealth - random.nextInt(3, 7); // Defender low health
decrease
15.
organisms[defendingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[defendingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2); // Give the defender an awareness
increase
16.         System.out.println(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and " + defendingOrganism.getOrganismId() + " has won");
17.     }

```

Java

The adaptation is shown above, with the speed, size and age all being separate factors, they can also be checked individually, which is seen in the IF / ELSE statement.

Testing this, the console below shows that organism 111 attacked organism 195, with organism 111 winning. To validate this, the factors from 111 and 195 should be compared. The stats for organism 111 should be superior to that of 195's, for this example to be correct.

File - Main

```

1 C:\Users\Harry\.jdks\jbr-17.0.9\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2023.2.5\lib\idea_rt.jar=51504:C:\Program Files\JetBrains\IntelliJ IDEA Community
Edition 2023.2.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 111 has attacked 195 and 111 has won

```

Below are the statistics for organism 111, highlighted in blue and underlined in red are the attributes of focus. The speed is 12.84, and the size is 0.4

```

"organismId":111,"yCoordinate":244,"stats":{"organismsMet":0,"thirdPreferredTerrainFood": {"waterBasedFood":true,"foodName":"Aquatic Plants","parentName":"Algae and tBred":63,"recentFood":[],"speed":12.84,"preferredTemp":8,"eatenFood": {"waterBased Name": "Water", "foodAvailability": 0.5, "landBasedFood": false}, "recentTemp":[], "start": 555,"health":0.0,"secondPreferredTerrain": "Default", "foodFactor": 0.2, "size": 0.4,

```

Below are the statistics for organism 195, highlighted in blue and underlined in red are the attributes of focus. The speed is 5.06 and the size is 0.3.

```

,"organismId":195,"yCoordinate":213,"stats":{"organismsMet":0,"thirdPreferredTerrain": "Default", "tsb": preferredFood": {"waterBasedFood":true,"foodName":"Aquatic Plants","parentName":"Algae and Aquatic Plant":false}, "ageLastBred":21,"recentFood":[],"speed":5.06,"preferredTemp":9,"eatenFood": {"waterBasedFood": 0.2,"terrainName": "Water", "foodAvailability": 0.5, "landBasedFood": false}, "recentTemp":[], "startSpeed": 2.5, "totalOrganismsMet": 216,"health":39.74,"secondPreferredTerrain": "Default", "foodFactor": 0.2, "size": 0.3

```

Considering the above results, it is clear that organism 111 should have won, due to its statistics being far superior to that of organism 195.

Another test will be conducted in the case that the defender wins. The layout will be the same, with the console log, JSON files, and an analysis.

2 attacks later within the same simulation, organism 153 attacked organism 310, with the defendant (organism 310) winning. For the predator prey cycle to be considered complete, organism 310's statistics must be superior to that of organism 153.

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.9\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.5\lib\idea_rt.jar=51504:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 111 has attacked 195 and 111 has won
3 151 has attacked 297 and 151 has won
4 153 has attacked 310 and 310 has won
```

Below are the statistics for organism 153, highlighted in blue and underlined in red are the attributes of focus. The speed is 6.48 and the size is 0.3.

Below are the statistics for organism 310, highlighted in blue and underlined in red are the attributes of focus. The speed is 5.39 and the size is 0.8.

This case is thankfully one of the more rarer cases, where the attributes have equal worth, with the size of the attacker being smaller, but the speed faster. In this case, the age should be also checked, with the younger organism winning. Organism 310 is 7 years old, as shown in the bottom of the image below. Organism 153 is 57. In this case, the situation is verifiable as correct due to the expected results being met. Organism 310 was younger than the attacker, despite having decreased speed, therefore was able to win the fight, despite suffering a health decrease. Organism 153 on the other hand, was an older organism and therefore had less of a chance against the other organism, losing the fight and dying (shown by the health of 0.0).

The predator-prey cycle is now complete with predator organisms seeking out and attacking others.

3.8 Saving and Loading Simulations

This is the final section of both front-end UI, and the link to the back-end JSON. This could not have been completed before 3.7 to ensure the data storage is final. Currently, the user has options to load and save simulations, however the buttons do nothing. To add functionality to these, the JSON for the simulation will be saved under either a user defined name, or the current date and time. The user will then be able to load this JSON before starting the game, which will load the exact simulation (benefits of JSON). This will require a larger JSON package, which will include global information (such as map seed and year) as well as all organisms, so the information can be sent into the program and displayed.

3.8.1 Saving the JSON

The first part is to ensure that the JSON can be saved correctly in a way which it can be read and passed back into the program when needed. Currently, the JSON only stores the organisms themselves, however it will also need to store attributes such as the year, map seed, and any other information which is linked to the current operation. For this, the saved JSON will look slightly different to what has been seen before. Below is the representation of how the JSON will be saved.

```

1. {
2.     "seed": -219.274572,
3.     "currentYear": 219,
4.     "windowWidth": 1920,
5.     "windowHeight": 1080,
6.     "predatorPreyEnabled": true,
7.     "naturalDisastersEnabled": false,
8.     "organisms": [
9.         {
10.             "xCoordinate": 271,
11.             "motherID": -1,
12.             "organismId": 0,
13.             "yCoordinate": 349,
14.             "stats": { /* Organism Statistics Omitted */ },
15.             "sex": 0,
16.             "currentTerrain": "Beach",
17.             "fatherID": -1
18.         }, // . . . Other Organisms Omitted
19.     ]
20. }

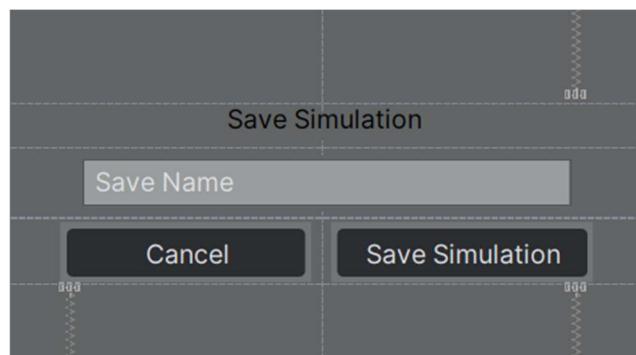
```

JSON

In the JSON above, key information such as the game seed, current year, window size and Boolean attributes are all key to replicating the game's state. If the user wishes to load a previous save, such as this one, the following process will happen. Similar to how the game is started at the moment, the seed will be sent to Perlin Noise, with the current year, window width and window height being sent to the game window, and the Boolean attributes (predator / prey cycles and natural disasters) sent to the organism's class. The organisms will replace the generate organisms call within game window, instead allowing the organisms to pass into the Organism class defined globally.

3.8.2 Implementing the ability to save

This section covers the method behind saving the JSON designed in 3.7.1, which will be called once the user saves the game. The user should have the ability to name the save file, however by default it will be set to the current date and time. The JSON file which is saved will be stored in a package locally, so that they are both separate from the other files and can be read from a constant location. Once again, for the UI I will use the IntelliJ forms creator, due to the ease of use. The form will ask the user for the file name, and the option to save or cancel, therefore does not have much to it. The design for this is below.



The text area 'Save Name' will be replaced with the date and time (default file name) and can be replaced by the user.

Adding functionality to this, the text input should contain the current date and time for which the default file name will be but will also allow the user to change that. Once the user clicks save, the text contained in the input field, if not empty, will be fetched and saved as the filename, containing the JSON which will also be generated. If the user decides to cancel, then the program will continue as normal without saving. See below for the code block implementing this, refer to the comments for explanations of specific sections.

```

1.    @Override
2.    @SuppressWarnings("unchecked")
3.    public void mouseClicked(MouseEvent e) {
4.        if(e.getSource() == saveSimulationButton){ // If the user chooses to save the
simulation
5.            String saveFileName = "./Save Games/" + saveNameTextField.getText() + ".json"; //
Set the path
6.
7.            JSONObject jsonObject = new JSONObject(); // The JSON which will be saved
8.
9.            jsonObject.put("seed", perlinNoise.getSeed()); // The current seed
10.           jsonObject.put("currentYear", GameWindow.years); // The current year
11.           jsonObject.put("windowHeight", GameWindow.HEIGHT); // The window height and width
12.           jsonObject.put("windowWidth", GameWindow.WIDTH);
13.           jsonObject.put("predatorPreyEnabled", false); // Whether predator prey is enabled
14.           jsonObject.put("naturalDisastersEnabled", false);
15.
16.           JSONArray jsonArray = getOrganisms(); // The array storing the organisms. Uses
getOrganisms() method
17.           jsonObject.put("organisms", jsonArray); // Add in the organisms to the JSON
18.
19.           // Write the JSON Object to the file
20.           try (FileWriter fileWriter = new FileWriter(saveFileName)) {
21.               fileWriter.write(jsonObject.toJSONString());
22.               fileWriter.flush();
23.           } catch (IOException error) {
24.               error.printStackTrace();
25.           }
26.       }
27.   }
28.
29.   if(e.getSource() == cancelButton){ // Else if they choose to cancel resume the game and
close the window
30.       GameWindow.gamePaused = false;
31.       frame.dispose();
32.   }
33. }
34.
35. // Returns a JSON array of all the organisms
36. @SuppressWarnings("unchecked")
37. private JSONArray getOrganisms(){
38.     JSONArray allOrganisms = new JSONArray();
39.
40.     for(Organism organism : GameWindow.organisms){ // Loops through each organism and
creates the JSON
41.         JSONObject newOrganism = new JSONObject();
42.         newOrganism.put("organismId", organism.getOrganismId());
43.         newOrganism.put("xCoordinate", organism.getxCoordinate());
44.         newOrganism.put("yCoordinate", organism.getyCoordinate());
45.         newOrganism.put("currentTerrain", organism.getCurrentTerrain());
46.
47.         JSONObject statsObject = getStatsObject(organism.getStats()); // Fetches the stats
of the current organism
48.
49.         newOrganism.put("stats", statsObject);
50.         newOrganism.put("sex", organism.getSex());
51.         newOrganism.put("fatherID", organism.getFatherID());
52.         newOrganism.put("motherID", organism.getMotherID());
53.         newOrganism.put("predator", organism.isPredator());
54.         allOrganisms.add(newOrganism); // Adds the current organism to the array of
organisms
55.     }
56.
57.     return allOrganisms;
58. }
59.
60. // Returns the stats of the organism as a JSON object
61. @SuppressWarnings("unchecked")
62. public JSONObject getStatsObject(Stats stats){
63.     JSONObject statsObject = new JSONObject();
64.

```

```

65.     statsObject.put("currentAwareness", stats.getCurrentAwareness());
66.     statsObject.put("organismsEncountered", stats.getOrganismsEncountered());
67.     statsObject.put("awarenessHistory", stats.getAwarenessHistory());
68.     statsObject.put("age", stats.getAge());
69.     statsObject.put("preferredFood", stats.getPreferredFood());
70.     statsObject.put("preferredFoodString", stats.getPreferredFoodString());
71.     statsObject.put("eatenFood", stats.getEatenFood());
72.     statsObject.put("recentFood", stats.getRecentFood());
73.     statsObject.put("foodFactor", stats.getFoodFactor());
74.     statsObject.put("sizeFactor", stats.getSizeFactor());
75.     statsObject.put("size", stats.getSize());
76.     statsObject.put("speed", stats.getSpeed());
77.     statsObject.put("startSpeed", stats.getStartSpeed());
78.     statsObject.put("currentTemp", stats.getCurrentTemp());
79.     statsObject.put("preferredTemp", stats.getPreferredTemp());
80.     statsObject.put("recentTemp", stats.getRecentTemp());
81.     statsObject.put("tempFactor", stats.getTempFactor());
82.     statsObject.put("preferredTerrain", stats.getPreferredTerrain());
83.     statsObject.put("secondPreferredTerrain", stats.getSecondPreferredTerrain());
84.     statsObject.put("thirdPreferredTerrain", stats.getThirdPreferredTerrain());
85.     statsObject.put("health", stats.getHealth());
86.     statsObject.put("dead", stats.isDead());
87.     statsObject.put("ageLastBred", stats.getAgeLastBred());
88.     statsObject.put("generation", stats.getGeneration());
89.     statsObject.put("organismsMet", stats.getOrganismsMet());
90.     statsObject.put("tsb", stats.getTSB());
91.
92.     return statsObject;
93. }

```

Java

This method replicates what is in the organisms.JSON currently, placing the array into a new object. The file name is saved as either the date and time, or a user selected name.

3.8.3 Testing the saved JSON

The saved JSON should replicate what was found in 3.7.1 and should also be under the filename of whatever was inserted into the text box. To test this, I will show the starting elements, and the process of saving the game, and then show the resultant saved JSON. See below.

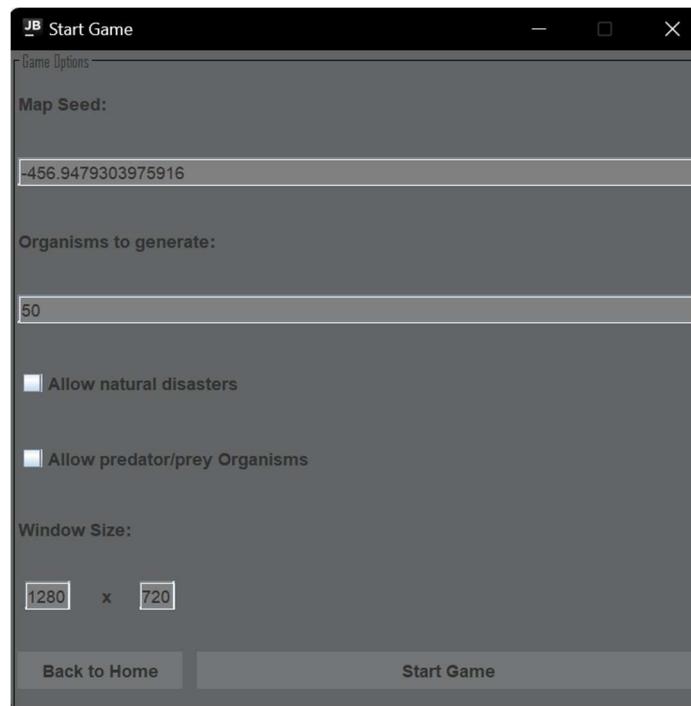


Figure 3.8.3 a – The seed, organisms to generate and window size



Figure 3.8.3 b – The saving method, including file name

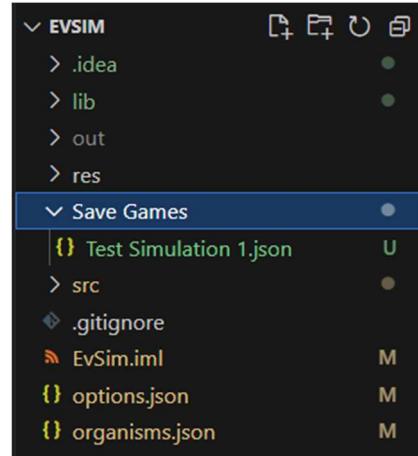


Figure 3.8.3 c – The saved JSON file, contained in Save Games directory under the name set in 3.8.3b

```

1. {
2.     "seed": -456.9479303975916,
3.     "predatorPreyEnabled": false,
4.     "windowHeight": 720,
5.     "organisms": [
6.         {
7.             "xCoordinate": 394,
8.             "motherID": -1,
9.             "organismID": 0,
10.            "yCoordinate": 541,
11.            "stats": {
12.                "organismsMet": 0,
13.                "thirdPreferredTerrain": "Default",
14.                "tsb": null,
15.                "preferredTerrain": "Default",
16.                "awarenessHistory": [
17.                    0.6
18.                ],
19.                "dead": false,
20.                "preferredFood": null,
21.                "ageLastBred": -1,
22.                "recentFood": [FoodTypes.FoodTypes@204c76ad],
23.                ],
24.                "speed": 5.05,
25.                "preferredTemp": 0,
26.                "eatenFood":FoodTypes.FoodTypes@44827ff2,
27.                "recentTemp": [
28.                    9
29.                ],
30.                "startSpeed": 5.0,
31.                "tempFactor": 1.0,
32.                "sizeFactor": 0.25,
33.                "generation": 1,
34.                "health": 98.75,

```

```

35.          "secondPreferredTerrain": "Default",
36.          "foodFactor": 0.25,
37.          "size": 0.5,
38.          "organismsEncountered": 0.0,
39.          "preferredFoodString": null,
40.          "currentTemp": 0,
41.          "age": 1,
42.          "currentAwareness": 0.6
43.        },
44.        "predator": false,
45.        "sex": 0,
46.        "currentTerrain": "Water",
47.        "fatherID": -1
48.      }, // . . . Other Organisms *
49.    ]
50.  }

```

JSON
* represents omitted information

While this worked, there are issues, such as the food types saving as an object token instead of the actual information needed. Furthermore, while this file is only saving 1 organism, if the user was to save a file when there are 1000+ organisms, the saving operation can be quite lengthy, due to the amount of reading and creating that has to happen.

3.8.4 Second Implementation

Therefore, instead of the original plan to save as one file, I am going to copy the organisms file into a created directory, which will contain 2 files. The first file has the information needed (years, seed, window size), and the second is a duplicated organisms.json. This means that long processing copying individual JSON factors is avoided, and the organisms.json can be copied and pasted back in order to load them. See below for this new method.

```

1.      String saveFileName = saveNameTextField.getText(); // The name of the folder and
JSON file
2.
3.      // File to be copied
4.      Path sourcePath = Paths.get("./organisms.json");
5.
6.      // Directory path for Save Games
7.      Path saveGamesDirectory = Paths.get("./Save Games");
8.
9.      // The path of the directory to create - ./Save Games/saveFileName
10.     Path destinationDirectory = saveGamesDirectory.resolve(saveFileName);
11.
12.    try {
13.      // Create the directory
14.      Files.createDirectories(destinationDirectory);
15.
16.      // The path of the file to create - ./Save Games/saveFileName/organisms.json
17.      Path finalDestination = destinationDirectory.resolve("organisms.json");
18.
19.      // Copies the ./organisms.json into ./Save Games/saveFileName/organisms.json
20.      Files.copy(sourcePath, finalDestination);
21.
22.      System.out.println("File copied successfully.");
23.    } catch (IOException error) {
24.      throw new RuntimeException(error);
25.    }
26.  }

```

Java

This takes advantage of the Files and Paths libraries to manipulate directories and files. As mentioned in the comments, different paths and directories are made before the organisms file is copied over. The resultant structure is shown to the right, with the organisms.json replicating what is found in the original organisms.json.

▼ Save Games	●
▼ 20231206_2137	●
{} organisms.json	U
▼ 20231206_2138	●
{} organisms.json	U

Now that the organisms.json is replicated, the second file can be created, containing the other information such as seed and year, as shown below.

```

1.     if(e.getSource() == saveSimulationButton) { // If the user chooses to save the
2.         simulation
3.             String saveFileName = saveNameTextField.getText(); // The name of the folder and
4.             // File to be copied
5.             Path sourcePath = Paths.get("./organisms.json");
6.
7.             // Directory path for Save Games
8.             Path saveGamesDirectory = Paths.get("./Save Games");
9.
10.            // The path of the directory to create - ./Save Games/saveFileName
11.            Path destinationDirectory = saveGamesDirectory.resolve(saveFileName);
12.
13.            // The second JSON file content containing the year, seed etc
14.            JSONObject jsonObject = new JSONObject(); // The JSON which will be saved
15.            jsonObject.put("seed", perlinNoise.getSeed()); // The current seed
16.            jsonObject.put("currentYear", GameWindow.years); // The current year
17.            jsonObject.put("windowHeight", GameWindow.HEIGHT); // The window height and width
18.            jsonObject.put("windowWidth", GameWindow.WIDTH);
19.            jsonObject.put("predatorPreyEnabled", false); // Whether predator prey is enabled
20.            jsonObject.put("naturalDisastersEnabled", false);
21.
22.        try {
23.            // Create the directory
24.            Files.createDirectories(destinationDirectory);
25.
26.            // The path of the file to create - ./Save Games/saveFileName/organisms.json
27.            Path finalDestination = destinationDirectory.resolve("organisms.json");
28.
29.            // Copies the ./organisms.json into ./Save Games/saveFileName/organisms.json
30.            Files.copy(sourcePath, finalDestination);
31.
32.            // Write the second JSON file
33.            FileWriter fileWriter = new FileWriter("./Save Games/" + saveFileName + "/" +
34.                saveFileName + ".json");
35.            fileWriter.write(jsonObject.toJSONString());
36.            fileWriter.flush();
37.
38.            frame.dispose(); // Removing the frame
39.            ErrorWindow successWindow = new ErrorWindow("<html><p style=\"text-
40. align:center;\">Simulation saved as:<br>" + saveFileName + "</p></html>"); // Displaying a success
41.            message
42.        } catch (IOException error) {
43.            throw new RuntimeException(error);
44.        }
45.    }

```

Java

3.8.5 Testing the second implementation

When the user chooses to save the simulation, they are provided with the options box shown in figure 3.8.5 a. If they select to save simulation, then the success dialogue is shown (figure 3.8.5b), and the relevant files are saved to the relevant locations (figure 3.8.5c).



Figure 3.8.5 a

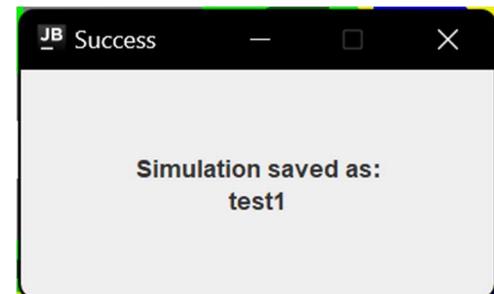


Figure 3.8.5 b

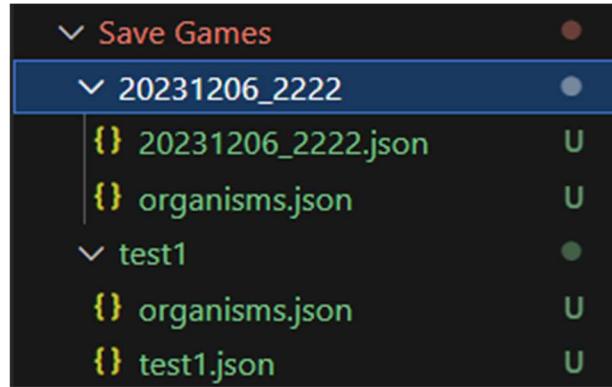


Figure 3.8.5 c

This demonstrates the success of the save state, meaning the user can now save simulations.

3.8.6 Loading the JSON Logic

For the JSON to be loaded, the opposite process must entail. The load option in both the main menu and in the game will have to display a list of save file names, as well as key information such as organism count and the year. This will help the user to determine the correct file that they want to load. The user should also have the option to delete and rename files.

Once the user selects a save file to load, the file should be read, and the relevant information passed into the program. As mentioned in 3.7.1, the seed will be sent to Perlin Noise, with the current year, window width and window height being sent to the game window, and the Boolean attributes (predator / prey cycles and natural disasters) sent to the organism's class. The organisms will replace the generate organisms call within game window, instead allowing the organisms to pass into the Organism class defined globally. During this process, the window will load as expected, containing the same information as how it was saved. There should be no difference between the game before the save, and the game which is loaded.

3.8.7 Implementing the ability to load the JSON

Once again I will be using the IntelliJ forms creator to display a list of save states to load. The design for this is shown below.

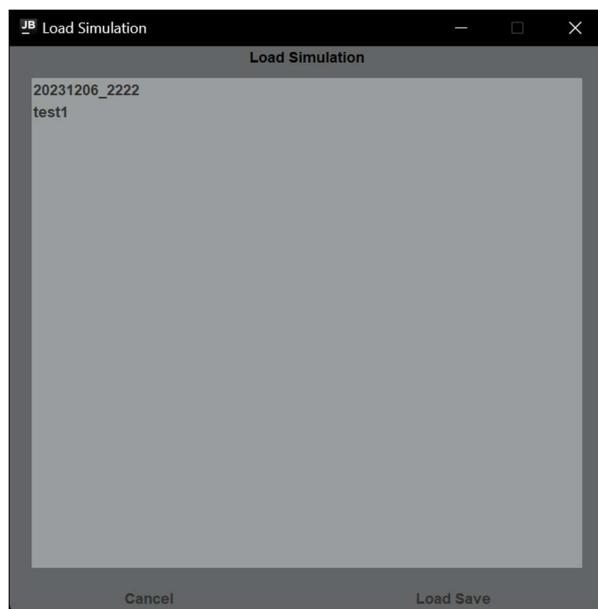


Figure 3.8.7 a

This window contains both the load and save buttons as well as a JList, which contains the names of the files. The list which is shown in figure 3.8.5a demonstrates what the list of saved files will look like. This can also be formatted to give the user more information such as when it was last modified. See below for the code implementing this as well as the result.

```

1.      // Path to the folder in which sub folders will be displayed
2.      String path = "./Save Games";
3.
4.      // Stores folder names
5.      DefaultListModel<String> listModel = new DefaultListModel<>();
6.
7.      // Returns list of folders
8.      File folder = new File(path);
9.      File[] folders = folder.listFiles(File::isDirectory);
10.
11.     // Loops through folders and adds
12.     if (folders != null) {
13.         for (File subFolder : folders) {
14.             // Send the last modified date and time to a variable which can be formatted
15.             LocalDateTime lastModifiedDateTime = LocalDateTime.ofInstant(
16.                 Instant.ofEpochMilli(subFolder.lastModified()),
17.                 ZoneId.systemDefault()
18.             );
19.             // Bullet points and formats date last modified
20.             listModel.addElement("<html><ul><li>" + subFolder.getName() + " Last Used: " +
lastModifiedDateTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy")) + "</li></ul></html>");
21.         }
22.     }
23.
24.     list1.setModel(listModel); // Add the model to the JList

```

Java

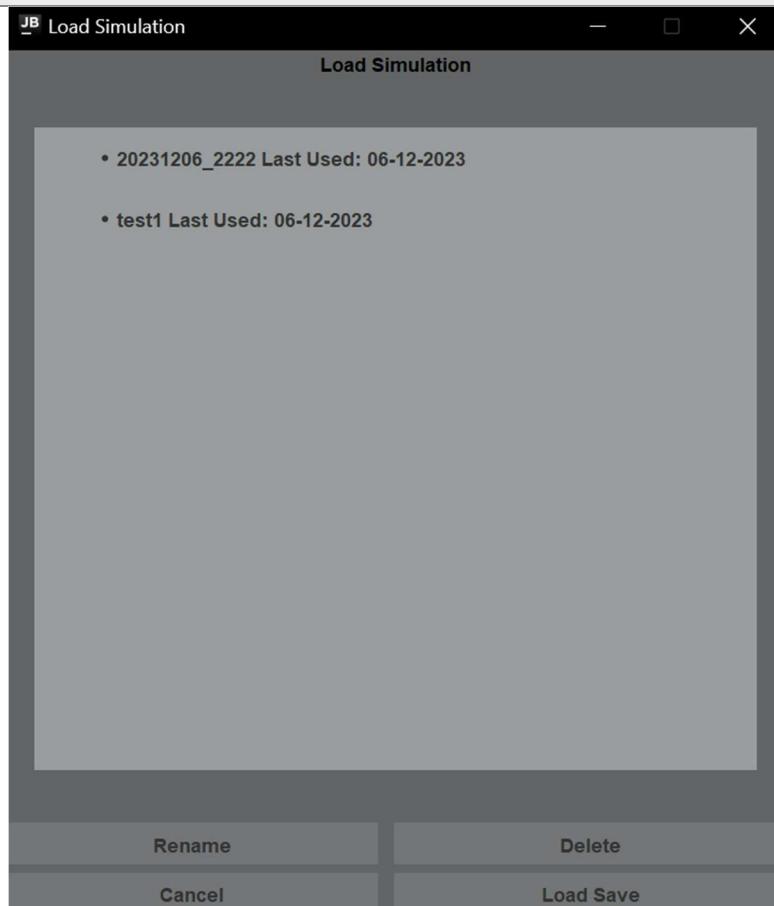


Figure 3.8.7 a – Final Result

Figure 3.8.5b shows how the save files will be listed to the user, with the date showing (to aid recognition), as well as the options to rename and delete saves. The exact layout and design of these buttons will change as functionality is added. Currently, the selected item from the JList is not used, however this will change. If the user selects an item from the JList, the rename, delete and load save buttons will become functional, allowing the user to do the corresponding action. The code block below shows the full implementation for this.

```

1.      @Override
2.      public void mouseClicked(MouseEvent e) {
3.          String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
4.
5.          if(e.getSource() == list1){ // If the user clicks on an object in the list
6.              renameButton.setEnabled(true); // Allow the other buttons to become functional
7.              deleteButton.setEnabled(true);
8.              loadSaveButton.setEnabled(true);
9.          }
10.
11.         // Load the simulation under the selected file name
12.         if(e.getSource() == loadSaveButton && loadSaveButton.isEnabled() &&
!currentlySelectedFile.isEmpty()){
13.             ObjectMapper objectMapper = new ObjectMapper();
14.             try {
15.                 // Replace organisms.json with the stored json file
16.                 Files.copy(Paths.get("./Save Games/" + currentlySelectedFile +
"/organisms.json"), Paths.get("./organisms.json"), StandardCopyOption.REPLACE_EXISTING);
17.                 // Load the stored settings
18.                 Settings newLoad = objectMapper.readValue(new File("./Save Games/" +
currentlySelectedFile + "/" + currentlySelectedFile + ".json"), Settings.class); // Creating an
array of organisms from organisms.json
19.                 // start a new game. Sends loaded save as true to indicate what to load. Sends
the newLoad to be loaded
20.                 GameWindow newGameWindow = new GameWindow(newLoad.getWindowWidth(),
newLoad.getWindowHeight(), true, newLoad);
21.                 newGameWindow.startGameThread(); // Start the thread
22.                 frame.dispose(); // Dispose of the load menu
23.                 OpeningMenu.frame.dispose(); // Dispose of the opening menu
24.             } catch (IOException ex) {
25.                 throw new RuntimeException(ex);
26.             }
27.         }
28.
29.         // Deletes a save (folder and JSON files)
30.         if(e.getSource() == deleteButton && deleteButton.isEnabled() &&
!currentlySelectedFile.isEmpty()){
31.             // Removes selected save from the list on the screen
32.             DefaultListModel<String> listModel = (DefaultListModel<String>) list1.getModel();
33.             listModel.remove(list1.getSelectedIndex());
34.
35.             // Deletes the folder itself
36.             try {
37.                 // Deletes the folder and its contents recursively
38.                 deleteFolder(Paths.get("./Save Games/" + currentlySelectedFile));
39.                 frame.dispose(); // Refresh the load save page
40.                 new LoadSimulation();
41.             } catch (IOException error) {
42.                 throw new RuntimeException(error);
43.             }
44.         }
45.
46.         // Renames a save file
47.         if(e.getSource() == renameButton && renameButton.isEnabled() &&
!currentlySelectedFile.isEmpty()){
48.             // Gets the name that the user wishes to, and renames in the class
renameFile
49.             new renameFile(currentlySelectedFile);
50.         }
51.

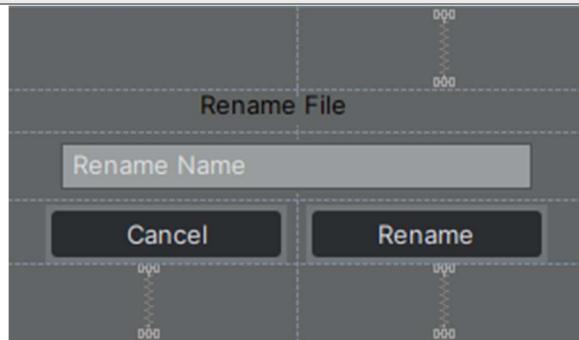
```

```

52.         // Close the frame
53.         if(e.getSource() == cancelButton) {
54.             frame.dispose();
55.         }
56.     }
57.
58.     // The folder cannot be deleted if it has contents in it, therefore this method is needed
59.     // to
60.     private static void deleteFolder(Path folderPath) throws IOException {
61.         // 'Walk' the file tree to delete each file starting from the given folder path
62.         Files.walk(folderPath).sorted((p1, p2) -> -p1.compareTo(p2)).forEach(path -> {
63.             try {
64.                 Files.delete(path); // Deletes the sub files/folders, in this case only
the sub-files
65.             } catch (IOException e) {
66.                 throw new RuntimeException(e);
67.             }
68.         });
69.     }
70.
71.     private String currentSelectedFile(){
72.         String currentlySelectedFile = "";
73.         String htmlString = list1.getSelectedValue(); // The selected value (which returns an
HTML string)
74.
75.         // Uses a pattern and a matcher to extract the name of the file from the HTML string
76.         Pattern pattern = Pattern.compile("<li>(.*)? Last Used:")); // Finds the file name
amongst the last name and HTML data
77.         Matcher matcher = pattern.matcher(htmlString);
78.
79.         if (matcher.find()) currentlySelectedFile = matcher.group(1); // If the pattern is
found then the file name can be updated
80.
81.         return currentlySelectedFile;
82.     }

```

Java



As seen on line 49, a rename file class is called. This class works almost identical to that of the save simulation, with the only difference being a rename instead of a create. The design is the same, with the wording changed, for example Save Simulation to Rename Save, see left.

The adjacent class for this takes the user's input, ensures it is not empty, and changes the file names.

```

1.     public renameFile(String originalFileName){ // Sends in the original file name to be set as
a placeholder
2.     currentlySelectedFile = originalFileName; // Stores the old string to be used to find
the file
3.     renameNameTextField.setText(originalFileName); // Sets the placeholder as the current
file name
4.
5.     frame = new JFrame("Rename File");
6.     frame.setPreferredSize(new Dimension(300,175));
7.     frame.add(panel1); // As the panel already contains the text, labels and buttons, only
the panel has to be added to the frame.
8.     frame.setResizable(false);
9.     frame.pack();
10.    frame.setLocationRelativeTo(null);
11.    frame.setVisible(true);
12.
13.    cancelButton.addMouseListener(this); // Add required mouse listeners
14.    renameFileButton.addMouseListener(this);
15. }
16.

```

```

17.     @Override
18.     public void mouseClicked(MouseEvent e) {
19.         // If they choose to rename the file, find the old files and rename appropriately
20.         if(e.getSource() == renameFileButton){
21.             String newFileName = renameNameTextField.getText(); // The new name
22.
23.             if(newFileName.isEmpty()) { // Ensure the user has not entered an empty string
24.                 new ErrorWindow("Please enter a name! It cannot be blank"); // Display an
appropriate message if so
25.             } else {
26.                 try { // Renames the file and folder
27.                     // First path is the json file containing seed and years, second is the
same file but what to name it as
28.                     Files.move(Paths.get("./Save Games/" + currentlySelectedFile + "/" +
currentlySelectedFile + ".json"), Paths.get("./Save Games/" + currentlySelectedFile + "/" +
newFileName + ".json"));
29.                     // First path is the folder, second is the new name for the folder
30.                     Files.move(Paths.get("./Save Games/" + currentlySelectedFile),
Paths.get("./Save Games/" + newFileName));
31.
32.                     LoadSimulation.frame.dispose(); // remove the old load page
33.                     frame.dispose(); // Remove the rename page
34.                     new LoadSimulation(); // Bring back the load page with the new name
35.                 } catch (IOException ex) {
36.                     throw new RuntimeException(ex);
37.                 }
38.             }
39.         }
40.         if(e.getSource() == cancelButton){
41.             frame.dispose(); // Remove the window
42.         }
43.     }

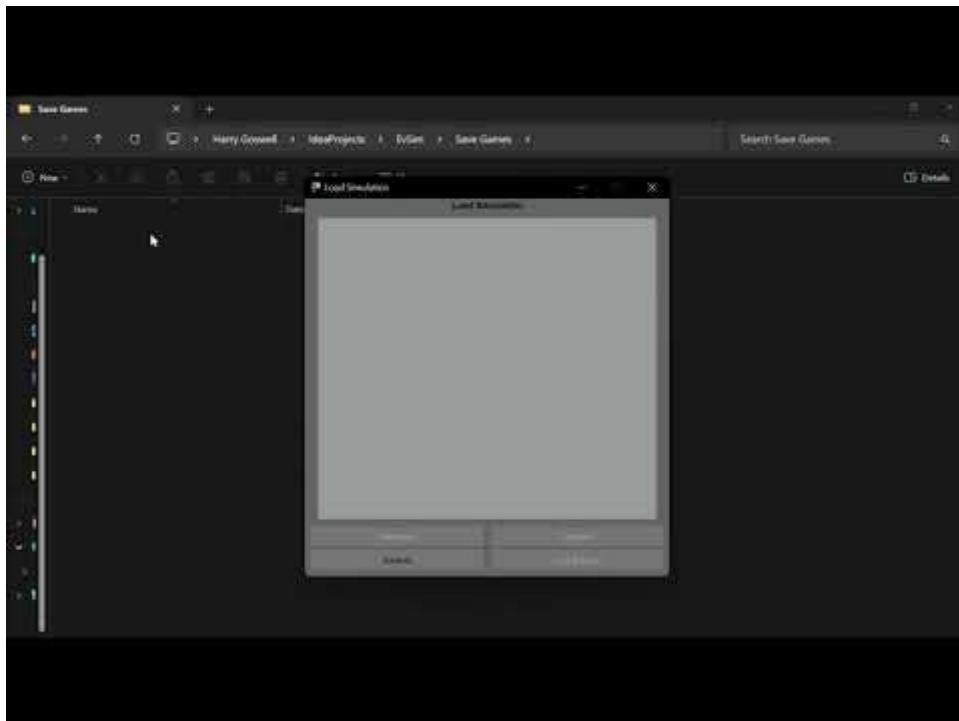
```

Java

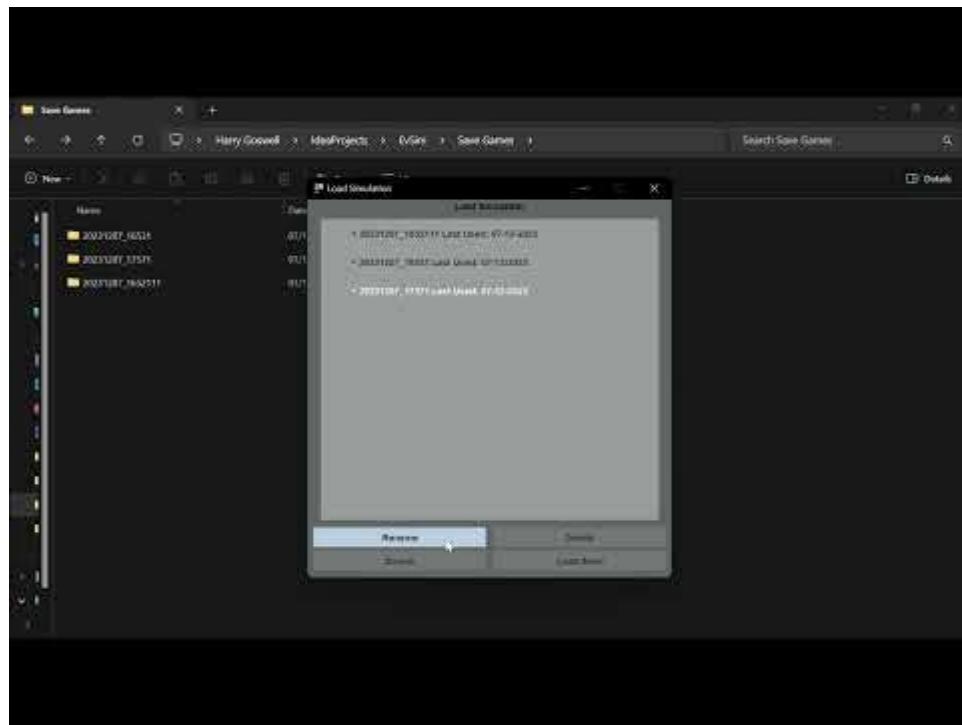
All code shown above completes the user's ability to load a previously saved program, with appropriate input validation and appropriate error messages if needed.

3.8.8 Testing the loaded JSON

For this to be successful, the loaded simulation should replicate what was saved, from the information on the side menu, to what is visible on screen. The options to load, rename, and delete should all function as intended also. The video below demonstrates the process of deleting files.



As the video shows, save states can be created and deleted efficiently. Below is a demonstration for the renaming of a file.



As before, the video shows the ability to rename saves and the effect it has on the physical save state. Loading in to a save state is shown below.



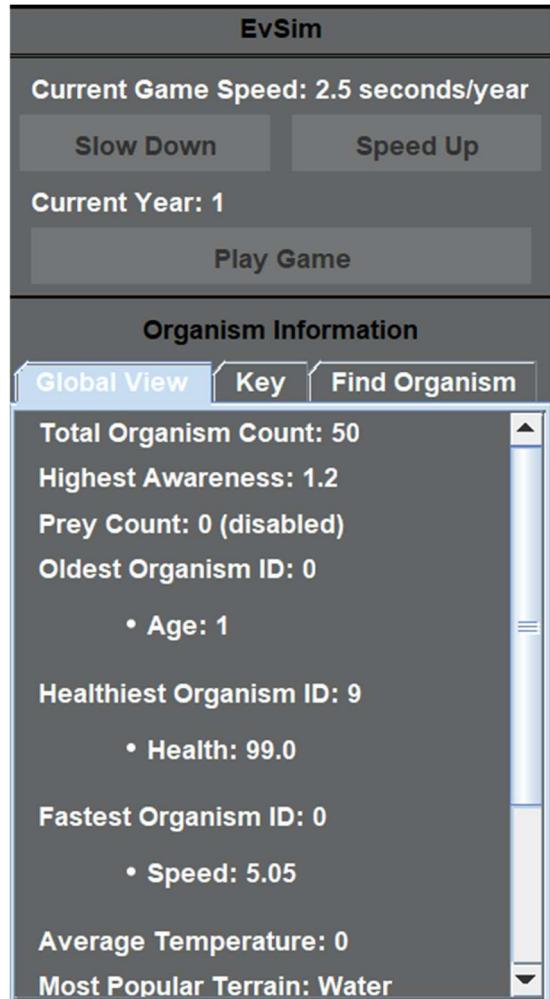
It can be established that all of the above work as expected, finalising the save state section.

3.9 Working on the user experience

This minor section will cover how the program has been checked to ensure that the user does not encounter any errors and is able to understand the program. This will include features such as an organism key, input validation and organism appearance.

3.9.1 Organism appearance

This is quite a key section of the success criteria and essential features. At the current stage, the organisms appear as small different coloured circles, meaning without proper explanation the simulation is useless to any user. Therefore I am going to change the colours and appearances of different organisms, alongside a key so that the user can understand the UI.



Using the redesigned side menu as seen to the left the user can get more information from the side menu without loosing any map space. With the organism information now within a scrollable panel, it takes up less vertical space too, allowing for smaller windows.

The key tab will show an image, which will demonstrate what the different colours mean. See figure 3.9.1a for the image within the key tab.



Figure 3.9.1 a

This means that the user can now easily identify the different types of organisms, for example the predator organisms, and the asexual organisms.

The Find Organism tab of the Organism Information section will allow the user to enter the ID of any organism, and have the information returned and the organism highlighted on the screen. See Figure 3.9.1b for how this section is designed.



Figure 3.9.1 b

To add functionality to this, the user input will be required to be validated, to ensure the user cannot enter any text letters, or invalid integers. If the user does input an invalid string, then an appropriate error message should be provided. See below for the method validating the string and providing the user with an appropriate error message.

```

1.         if(mouseEvent.getSource() == findButton){ // If the find button was pressed
2.             try {
3.                 GameWindow.organismSelected =
4.                     Integer.parseInt(organismToFind.getText().trim()); // Validate and set the input
5.                     organismInfoTab.setSelectedIndex(0); // Return to organism information tab
6.                     organismToFind.setText(""); // Clear the text from the text box
7.             } catch (NumberFormatException e) { // If it could not be validated create the
error message
8.                 new ErrorWindow("Organism not found!");
9.             }

```

Java

This section attempts to pass the input into an integer, if that returns an error then the user is presented with the error window, with the help message saying the organism was not found. If the organism is found, then the text box is reset, and the tab switches to organism information which shows the information needed.

3.9.2 Side menu information

This section covers how the side menu will display the key organism information to the user. Principally, this section is finalising what was designed in 3.2.4 Displaying Stats to the User. The information which I will choose to display for the global information are shown below.



Figure 3.9.2 a

The global statistics are now represented in a much cleaner format. Instead of bullet pointed information, it is laid out in way which shows the organisms ID (if applicable) and the statistic itself underlined next to it. This once again reduces the size taken up by the statistics, removing the need for a scrollable panel.

The information I am going to show for specific organisms will be similar, based off of the design in 3.2.4. See below for the new design.

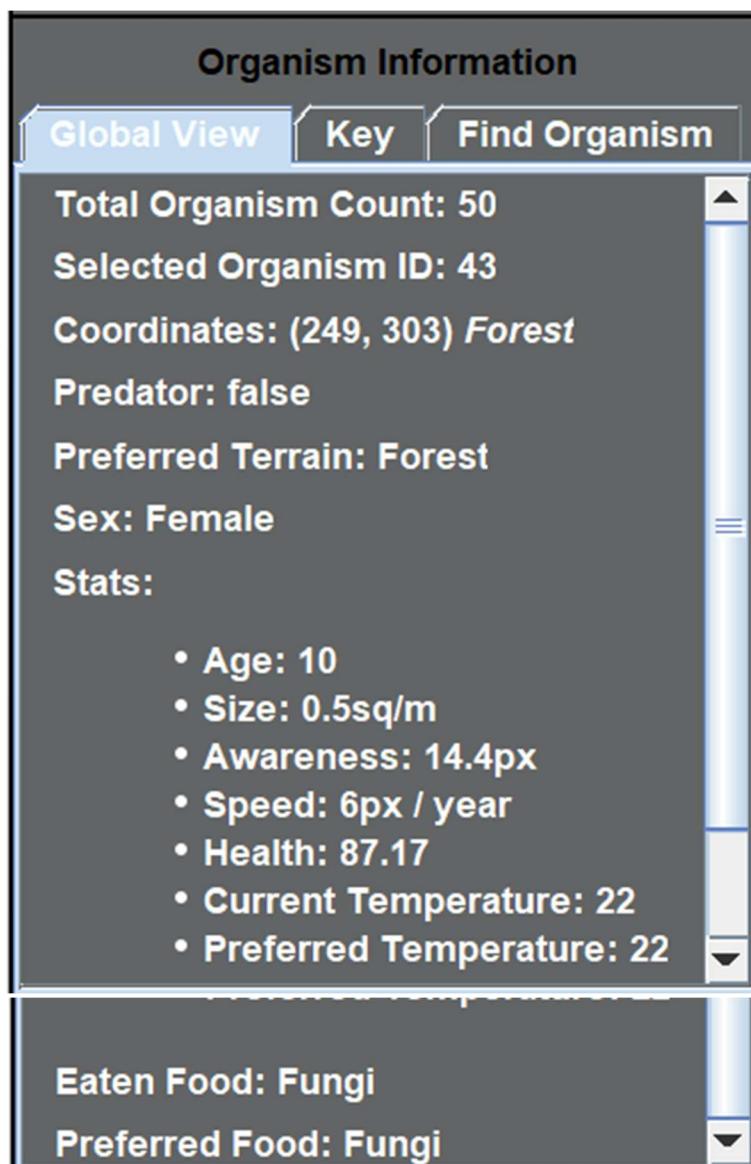


Figure 3.9.2 b

Due to how many stats have to be displayed to the user here, this section does require the use of a scroll pane. The only missing information at first sight is the preferred and eaten food. The Global View section now provides a detailed and accurate insight into the global and singular stats for organisms, benefitting the user and allowing them to evaluate organisms.

3.9.3 Allowing the user to prevent predator/prey cycles



On the start menu, the user is presented with the option to disable predator / prey organisms, as shown in the image to the left. If the user unchecks this, nothing happens. To counter this, then Organisms class will

contain a Boolean variable, which can be set to mirror what this box was checked as, similar to how the 'Show Dead Organisms' checkbox functions within 3.9.1. By default, this option will be marked as true, as without the implementation of predator/prey organisms, there will be more organisms overall, and a less balanced simulation.

```
1. Organisms.predatorPreyEnabled = allowPredatorPreyOrganismsCheckBox.isSelected();  
Java  
Contained within startGameMenu.java
```

```
1. if (!organisms[this.organismID].isPredator() && predatorPreyEnabled) checkPredation();  
// Checks if the organism should be a predator  
Java  
Contained within Organisms.java
```

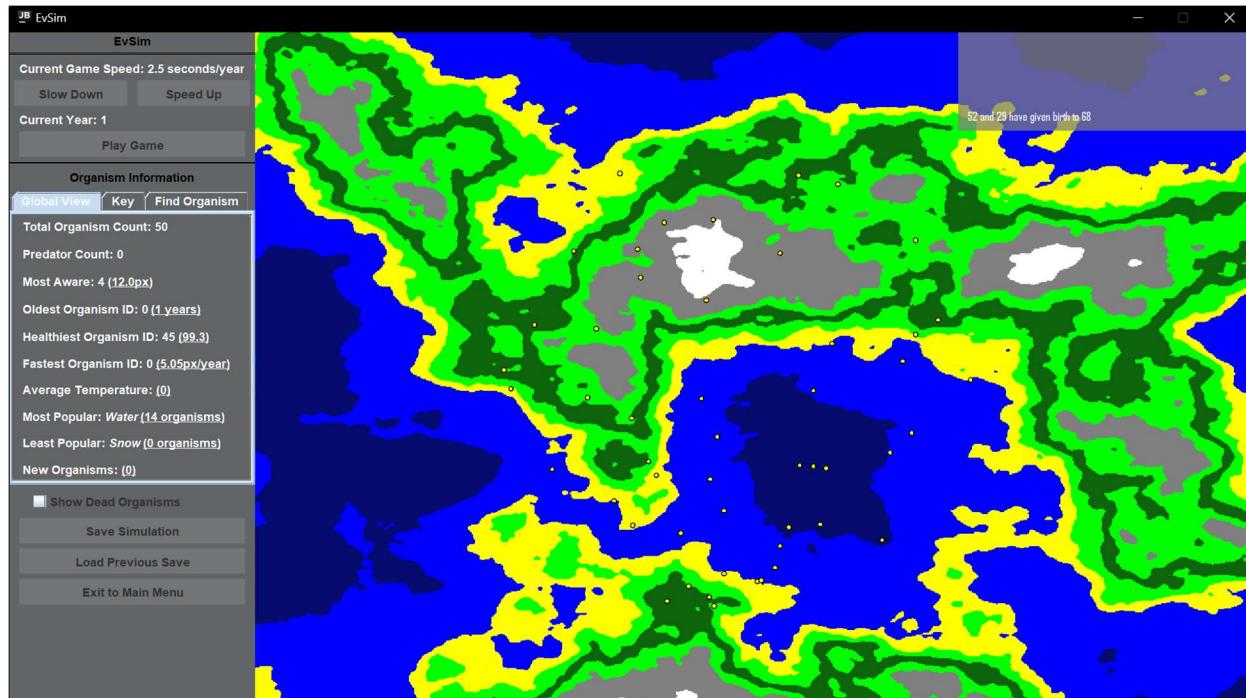
The two sections of code above ensure that the predator/prey can be disabled. The second code block does not allow an organism to be checked for predation if it is not enabled. If organisms cannot evolve to become a predator, then the predator/prey cycles are disabled. This can then also be transferred over to the physical save file once the user decides to save.

```
1. jsonObject.put("predatorPreyEnabled", Organisms.predatorPreyEnabled); // Whether predator prey  
is enabled  
Java  
Contained within SaveSimulation.java
```

Another section which has been untouched since the start of development, are natural disasters. For the full section on this, refer to 3.10.

3.9.4 Console log

This section will implement a console feature within EvSim, which will display to the user any events that happen, for example births, deaths, organisms becoming asexual, organisms becoming predators, etc. Giving this information to the user allows for a much larger scope of vision across the program, potentially allowing the user to conduct deeper hypotheses.



In this test example above, the grey box in the top right of the screen shows what an ideal design would be for the console. This example was drawn using the Graphics component, meaning it is not functional.

To implement the console itself, I will need to use a combination of a JPanel, JTextArea and a JScrollPane. Each of these can be altered as needed, for example the text area to add in console messages, and the scroll pane to allow preview of old messages. This should be able to be removed and added by the user, in the event that the panel covers up components behind or is simply not required by the user. For this, another checkbox will be added to the side menu, offering the user the ability to show or hide console, by default this will be off. This means that the console does not have to be created or added until the user selects the checkbox. For this reason, the creation of the console will reside mainly in the paintComponent section of the game window. The only downside to this being it will not update until a year has passed, potentially prolonging the console is show on screen for. Due to their being no workaround for this (how time works within EvSim), it is unfortunately not correctable. The console components will be defined as shown below.

```
1. private final JTextArea consoleTextArea = new JTextArea();
2. private final JScrollPane consoleScrollPane = new JScrollPane(consoleTextArea);
3. public static JPanel consolePanel = new JPanel(new BorderLayout());
```

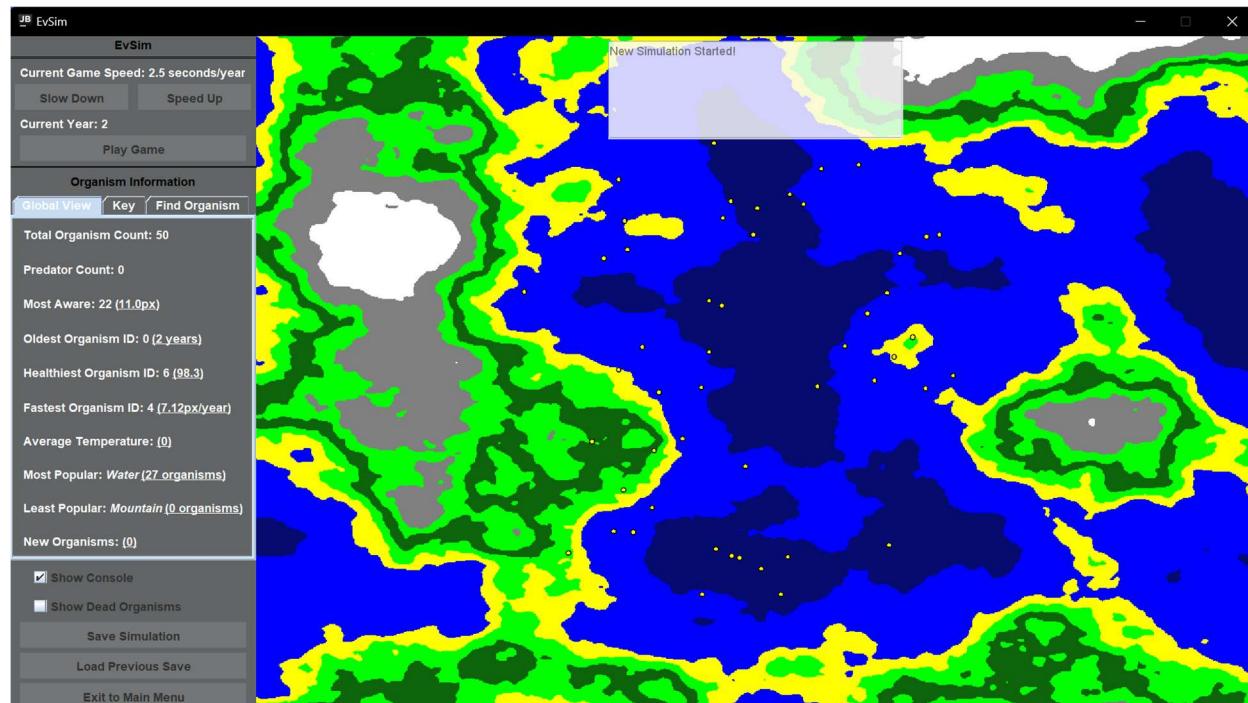
Java

In this, the scroll pane becomes a scroll pane for the text area, and the console panel is initialised with a border layout (so it does not cover the whole page). The console panel can then be added as follows.

```
1. if(!showConsole) remove(consolePanel); // If the console is not desired to be shown,
remove it
2. else { // Else, add it
3.     consolePanel.add(consoleScrollPane); // Add the scroll pane (which contains the text
area)
4.     consolePanel.setPreferredSize(new Dimension(300, 100)); // Set the size
5.     add(consolePanel); // Add it to the frame
6. }
```

Java

Due to the layout of the map, and lack of original plans to contain a console, the console here can only be added to the centre middle of the screen. Thankfully, due to the use of transparency, the console still shows what is behind it, avoiding the potential for it to hide the simulation itself. The image below shows what the console looks like, with a message to the user once they have started a new simulation.



To finish the log, is to add what is going to be displayed. The things that I am going to display in the log, are mainly the evolutionary adaptations, as these are rarer and will not fill the console.

The console can be added to through a `consoleLog` method, which adds a string to the text area.

```
1.     public static void consoleLog(String message) { // Takes a string input and adds to the
        console
2.         consoleTextArea.append(message + "\n"); // Add to the text area, and allow for the next
line to be added to through \n
3.         SwingUtilities.invokeLater(() -> { // Sets the scroll bar to be at the bottom to always
show most recent messages
4.             JScrollPane verticalScrollBar = consoleScrollPane.getVerticalScrollBar();
5.             verticalScrollBar.setValue(verticalScrollBar.getMaximum());
6.         });
7.     }
```

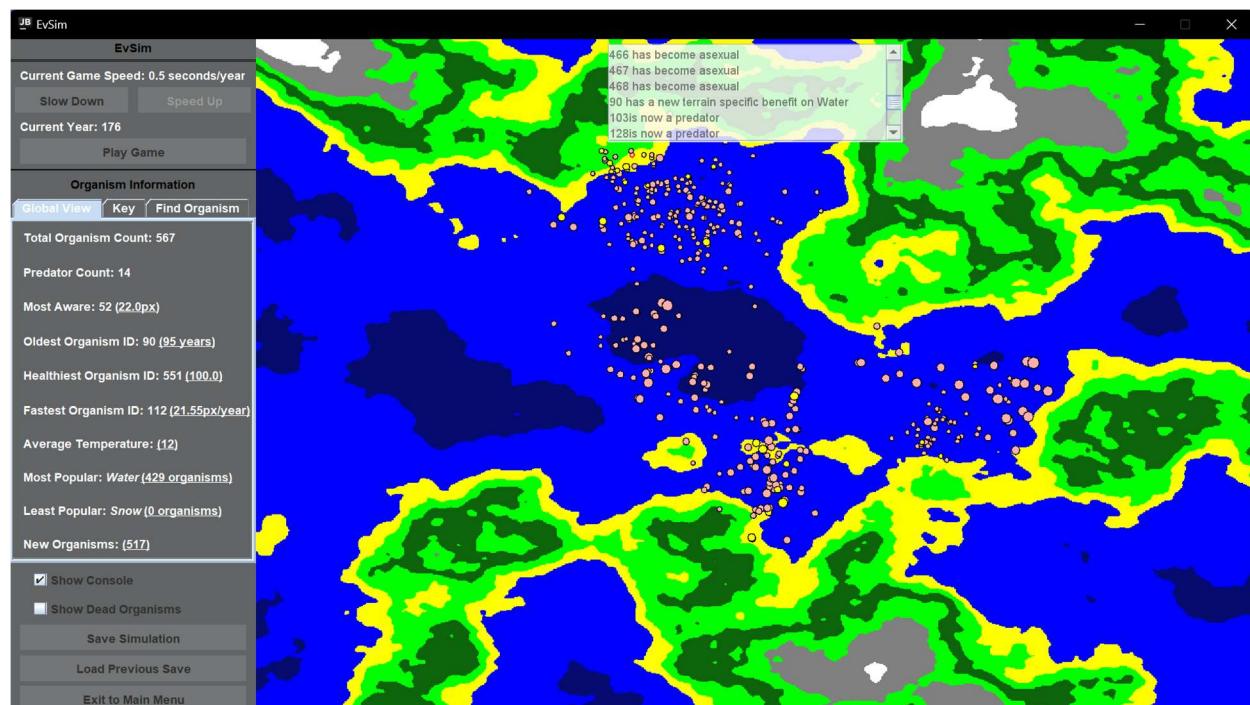
Java

This can be accessed through Organisms once an organism develops an evolutionary adaptation, as shown below.

```
1.                 GameWindow.consoleLog(currentOrganism.getOrganismId() + " has a new terrain
specific benefit on " + stats.getTSB());
```

Java

Now that the console can be accessed and read to, after 176 years of a simulation, this is how the console appears to the user.



A major benefit of using the console in this way is that if the user interacts with the console, or something new is added to the console, it highlights away from being transparent, significantly increasing both the readability of the console (when it is highlighted) and the use of the program when it is not highlighted. One issue in the image above is the lack of space between the organism ID and message for an organism becoming a predator. This was a simple fix, as the issue was a missing string space.

3.9.5 Balancing evolution

This section will be considerably hard to achieve, as balancing something which is not measurable can prove to be a tricky task. In terms of balancing EvSim, I am going to focus on a balanced simulation where there are not too many organisms, for example if the number of alive organisms exceeds 500, the simulation will struggle to execute each year, becoming heavily dependent on system resources. This is why balancing is required, as with a balanced amount of organisms the system is less likely to struggle at greater organism counts, improving the user experience.

Due to this section being largely a lot of trial and error, I will document the changes that were made, including the original values, and a justification as to why the change was made.

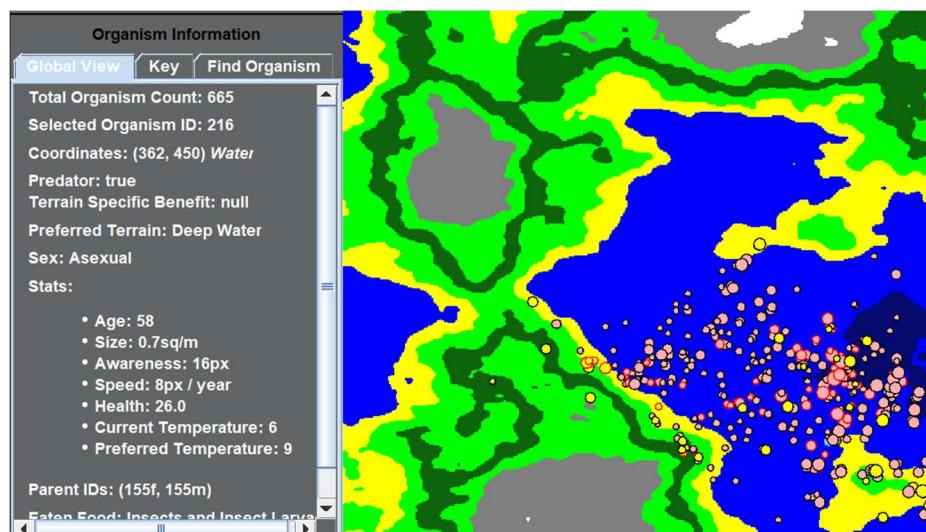
3.9.5.1 Predator Prey Adjustments

The current criteria for an organism to become a predator, is 2 previous generations of father organisms which have had 500 or more organism interactions. Overall, it is very rare for organisms to become a predator, however the adjustment made should only be small, as it only takes a small change to have too many adapting. The other issue currently with predator / prey cycles is that the moment an organism becomes a predator, its health declines so rapidly that it has a minimal affect on the total organism population – adverse to the intended affect. This means the health hit that the predator organisms take should be much lower, with the defending organism taking the same amount. The justification behind this being the incredibly large amount of organisms which are born once the year passes ~100. To counter the amount of organisms alive, the predator organisms should attack and therefore kill off a large portion of organisms, keeping the number of alive organisms balanced and not overcrowded.

Predator: true
Preferred Terrain: Grass
Sex: Male
Stats:
<ul style="list-style-type: none"> • Age: 18 • Size: 0.8sq/m • Awareness: 20px • Speed: 4px / year • Health: 21.27

As shown by the screenshot on the left, organisms which become predators die incredibly fast, with this organism's age being only 18, with a health of 21.27. Ideally, predators should live the same length of time as other organisms, only a little shorter due to their tendencies to attack other organisms.

The changes I have made are as follows. The default health decrease for both organisms (originally 5) has been removed. The amount of health the predator organism loses has been decreased to 1-4 for a win (originally 3-7 + default decrease) and 4-8 (originally 7-13 + decrease) for a loss. The organism can only attack every 15 years after the age of 10 (originally every 10 from birth). The organism can now become a predator if 2 generations of father organisms have had over 450 encounters each (originally 500 each).

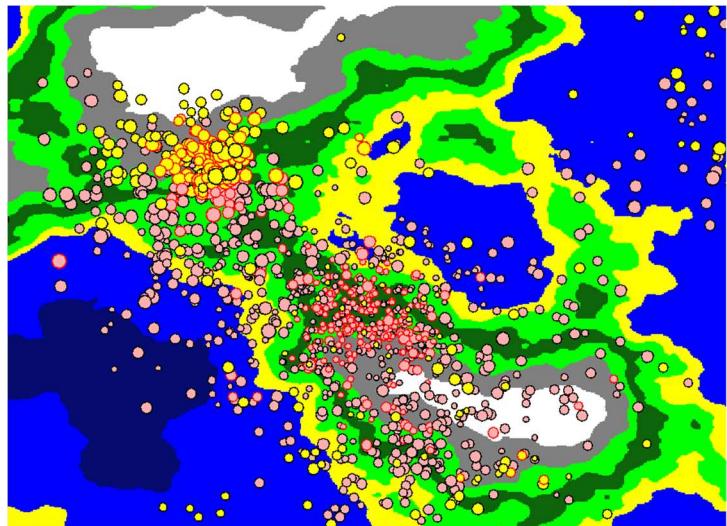


Due to these changes being so great, the difference it has made is also great. As shown by the organism statistics on the left, the organism is a predator, and is 58 years old with a health of 26.0. A significant improvement from the previous example. Furthermore, it is clear from the organism cluster on the right side of the image above that it is much more common for organisms to become a predator, achieving the intended result.

3.9.5.2 Breeding

The main issue with organism breeding is the rate at which it happens. Despite there being a slow start, after 200 organisms the number of organisms increases exponentially, faster than predators can attack. This is mainly due to the low criteria that organisms have to breed with each other. For example, a male or female organism can breed every 5 years for its whole life after the age of 15, meaning for every 2 organisms there is one more every fifth year. This can continue for as long as 80 years.

Evidently, the breeding criteria needs to be changed to bring down the amount of organisms which breed. To prevent this, I will increase the age that organisms can breed, and also the age difference between them, *once the number of organisms reaches a certain amount*. If an organism is 30 years older or younger than the other organism, it cannot breed. The cooldown time for organisms breeding will also be increased to every 10 years after 100 organisms and 20 after 150.



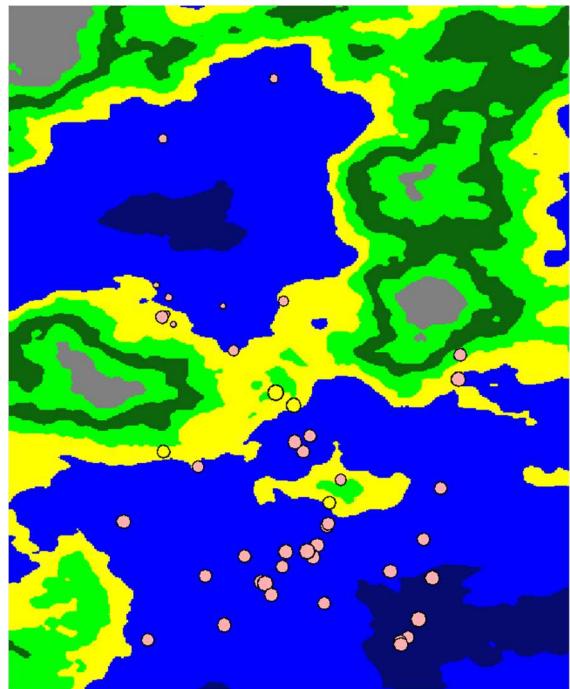
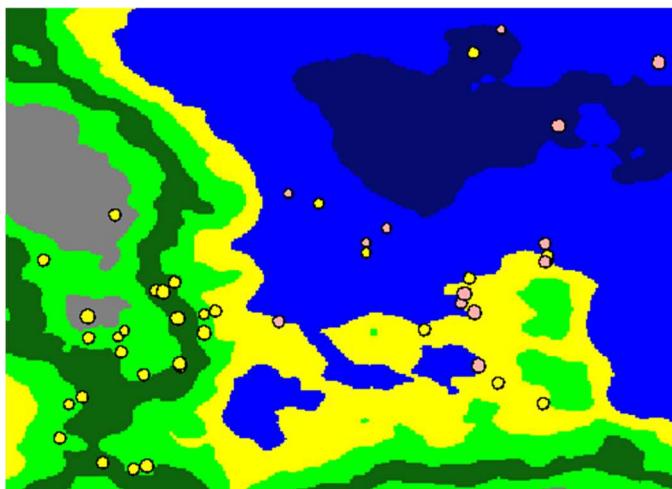
The new criteria for organisms is as follows. Note that the nested if statements are required due to how large the criteria is.

```
1.     int defaultAgeBarrier = 5;
2.     if(organisms.length > 150) defaultAgeBarrier = 20; // The amount of time between
organisms giving birth
3.     else if (organisms.length > 100) defaultAgeBarrier = 10;
4.
5.     if (originOrganism.getSex() != secondaryOrganism.getSex() && 30 >
Math.abs(originOrganism.getStats().getAge() - secondaryOrganism.getStats().getAge())) { // If they
are male and female
6.         if (originOrganism.getStats().getAgeLastBred() < originOrganism.getStats().getAge()
- defaultAgeBarrier && secondaryOrganism.getStats().getAgeLastBred() <
secondaryOrganism.getStats().getAge() - defaultAgeBarrier) { // If both are older than 15
7.             if (Math.abs(originOrganism.getxCoordinate() -
secondaryOrganism.getxCoordinate()) <= 5 && Math.abs(originOrganism.getyCoordinate() -
secondaryOrganism.getyCoordinate()) <= 5) { // If they are within a radius of 5
8.                 if (originOrganism.getSex() == 1) {
9.                     createChildOrganism(originOrganism, secondaryOrganism);
10.                    originOrganism.getStats().setAgeLastBred(stats.getAge());
11.                    secondaryOrganism.getStats().setAgeLastBred(stats.getAge());
12.                } else if (originOrganism.getSex() == 0) {
13.                    createChildOrganism(secondaryOrganism, originOrganism);
14.                    secondaryOrganism.getStats().setAgeLastBred(stats.getAge());
15.                    originOrganism.getStats().setAgeLastBred(stats.getAge());
16.                }
17.            }
18.        }
19.    }
```

Java

The second issue currently standing with breeding cycles is the amount of organisms becoming asexual. While it is a good thing that asexual organisms exist and breed, it can happen too often. After running many simulations, asexual organisms are either overpopulated, or underpopulated, depending on the terrain around. This means that any major change made could have a significant adverse affect. To change the amount of organisms which become asexual, I am going to decrease the amount of organisms met by the mother organisms to 5, instead of 10.

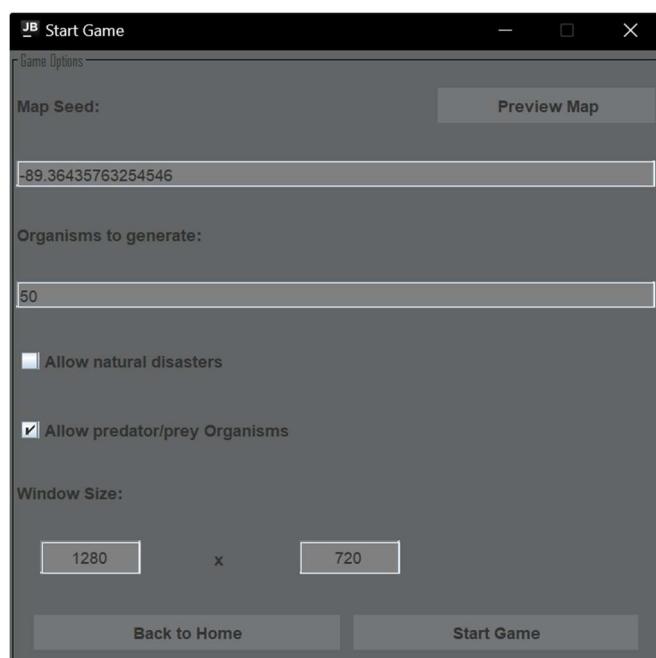
In the two examples shown above, the proportion of organisms after 300 years is much more refined. There are no longer large crowds of organisms, with a balance between asexual and sexual organisms. Note that the example to the right is heavily balanced towards asexual organisms dominating due to the amount of water at the epicentre of this seed. On a seed with less water at the epicentre, the balance is much more evenly spread, as seen below.



This is a major improvement compared to previous simulations, with organisms now not clustering as much, and a more even spread of organism types.

3.9.6 Map preview

This is another small feature which can prove to be quite beneficial for the user. Currently, the map seed which the user can set at the start of the simulation is not something which the user will know how to interact with. By showing or having an option to show how the map will look depending on the different seed will add use to the feature. By allowing a preview of the map also, the user will not have to consistently create new simulations to find a map that suits their criteria. For this to be added, the start menu will contain a preview map button, which once pressed will generate the map only, alongside the option to return to the start menu to change the seed/other attributes. The new design for the start menu is shown below.



A few things were changed here also, as there were a few discrepancies with the layout, for example the size of the buttons along the bottom row being disproportional and the window size sometimes not expanding enough. Due to having more experience with the IntelliJ forms creator now, I was able to correct these issues by surrounding them with a JPanel and setting preferred sizes. The preview map button at the top right is now also available to the user, with the added logic shown below.

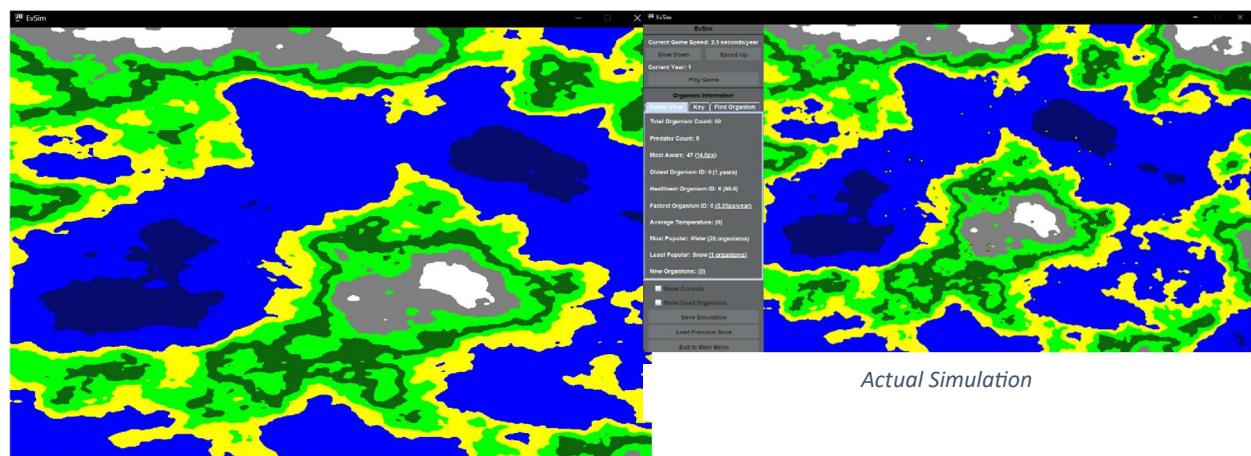
```

1. // . . . Other code
2. if(e.getSource() == previewButton){
3.     double seedInput = Double.parseDouble(seedTextField.getText()); // Parses the data
from seedTextField into a double
4.     if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
5.         int width = Integer.parseInt(a1920TextField.getText()), height =
Integer.parseInt(a1080TextField.getText());
6.         PreviewMap(width, height);
7.     } else {
8.         // If it is not in the bounds, then it displays the ErrorWindow, with html
styled text
9.         new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a seed
that is within the range of -1000.0 and 1000.0</p></html>");
10.    }
11. }
12. }
13.
14. public void PreviewMap(int width, int height) {
15.     JFrame frame = new JFrame("EvSim");
16.     frame.setPreferredSize(new Dimension(width - 250, height));
17.     frame.setLocation((int) (OpeningMenu.screenSize.getWidth() / 2) - width / 2, 0);
18.     frame.setResizable(false);
19.
20.     perlinNoise.setSeed(Double.parseDouble(seedTextField.getText()));
21.     generateMap mapPanel = new generateMap(width + 250, height, false, null);
22.     frame.add(mapPanel);
23.
24.     frame.pack();
25.     frame.setVisible(true);
26. }
27. // . . . Other code

```

Java

The map preview creates a new window, with the same seed as what was input by the user, allowing for a preview of what the map could look like.



Preview 1

3.10 Natural Disasters / Seasons

Due to the way that the map / other factors contribute to the processing of EvSim, to create natural disasters within the simulation would be a complexity beyond what the nature of EvSim demands. Modifications to the map, evolution and other factors requires a lot of time and work. Due to this, I am

changing the implementation of natural disasters into the implementation of seasons within EvSim. In many ways, seasons are better to implement into EvSim as opposed to natural disasters, due to seasons being present everywhere in real life, whereas natural disasters only in select places. Furthermore, due to EvSim currently having plentiful algorithms throughout, natural disasters would not contribute any more to the success criteria / essential features than seasons would. There is no explicit mention of natural disasters throughout the essential features and success criteria, with the first mention in design. After conference with the stakeholders as well, it is not an essential feature on their behalf and therefore the change from natural disasters to seasons is justified.

3.10.1 Logic

The logic for seasons will work much like real life. However, due to the fact that the simulation runs year on year, there cannot be all four seasons in one year. To counteract this, the season will change every year, meaning every cycle lasts 4 years. The simulation will start on a random season, and will follow suit similar to real life: Winter → Spring → Summer → Autumn. Each season will have a temperature affect on each terrain, for example during winter all terrains will see a decrease in temperature between 8 and 10 degrees, whereas in summer, the temperature will increase between 8 and 10 degrees respectively. Due to this, only the temperature section of Organisms has to see a change. The season will be stored as a string, and will be checked at the beginning of each temperature check. After all organisms have been iterated through (year has gone by), the year will be checked to see if it is a factor of 4, if it is then the season is changed. The string can also be displayed to the user on the screen at the top right, avoiding overuse of the side menu.

3.10.2 Implementation

This section covers the process of implementing seasons into EvSim, from the iteration of the season to the display on the screen.

3.10.2.1 Iterating seasons

The season's iteration will be implemented as follows. As mentioned in the logic, the year will be checked if it is a factor of 4, if it is then the season will increment respectively. The code block below carries out this process.

```

1.    // This method checks the current season and increments if applicable.
2.    // If the user has chosen to turn off seasons then this method is not called.
3.    private void updateSeason(){
4.        if(GameWindow.years == 0){ // If it is the beginning, select a random season
5.            season = switch (random.nextInt(0, 4)) { // Random season
6.                case 0 -> "Winter";
7.                case 1 -> "Spring";
8.                case 2 -> "Summer";
9.                default -> "Autumn";
10.           };
11.       }
12.       if(GameWindow.years % 4 == 0 && GameWindow.years != 0){ // Update the season every 4
years
13.           season = switch (season){
14.               case "Winter" -> "Spring";
15.               case "Spring" -> "Summer";
16.               case "Summer" -> "Autumn";
17.               default -> "Winter";
18.           };
19.       }
20.       System.out.println(season); // Check if the intended result is achieved
21.   }

```

Java

The test for this is also shown below. For the test to be successful, the console should show a random season chosen at the start, followed by the same season for 4 years, before iterating to the next season. An ideal console result would be: Summer, Summer, Summer, Summer, Autumn, Autumn...

File - Main

```
1 C:\Users\Harry\.jdks\jbr-17.0.9\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.5\lib\idea_rt.jar=56109:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Harry\IdeaProjects\EvSim\out\production\EvSim;C:\Users\Harry\Downloads\json-simple-1.1.1.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-databind-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-annotations-2.13.3.jar;C:\Users\Harry\IdeaProjects\EvSim\lib\jackson-core-2.13.3.jar Main
2 Winter
3 Winter
4 Winter
5 Winter
6 Spring
7 Spring
8 Spring
9 Spring
10 Summer
11 Summer
12 Summer
13 Summer
14 Autumn
15 Autumn
16 Autumn
17 Autumn
18 Winter
```

The above console demonstrates a full year cycle as intended, starting from winter, and every four years a new season is began.

3.10.2.2 Giving the seasons an effect

This section covers the impact seasons have on the simulation, as defined in the logic. The following temperature impact will take place each year.

Winter -> Temperature decrease between 8 and 10 degrees.

Spring -> Temperature decrease between 3 and 8 degrees.

Summer -> Temperature increase between 8 and 10 degrees.

Autumn -> Temperature increase between 3 and 8 degrees.

The implementation of this into a code block is shown below.

```
1.    // This method updates the organisms preferred temperature as well as the temperature
2.    // factor
3.    // Takes in the current temperature of the terrain, as well as the preferred temp (if
4.    // applicable)
5.    // Is affected by the current season (if enabled) See 3.10 Documentation
6.    private void updateTemperature(TerrainAttributes terrainAttributes) {
7.        int temperatureChange = switch (season) { // The temperature change which will take
8.            place due to the season
9.                case "Winter" -> random.nextInt(-11, -7); // Bound is exclusive so set to -7
10.               case "Spring" -> random.nextInt(-8, -2);
11.               case "Summer" -> random.nextInt(8, 11);
12.               default -> random.nextInt(3, 9);
13.           };
14.       // . . . Other code
```

Java

The temperature change is then added onto the new temp, and the rest of the code is executed as normal. The test for this section will be carried out at the end of this section (see 3.10.3).

3.10.2.3 Displaying this on the screen

Due to this not being a factor displayed on the side menu, the season will instead be shown on the top right of the screen, as plain text. Due to the size of the string being entirely dependant on the length of the word, the position of the text will have to be variable. See below for how the season is going to be displayed.

```
1.    public void paintComponent(Graphics g) {
2.        super.paintComponent(g);
```

```

3.     g.setFont(new Font("Agency FB", Font.BOLD, 20)); // Custom font and ideal size
4.     g.setColor(Color.BLACK);
5.     FontMetrics fontMetrics = g.getFontMetrics(); // Used to get the width of the text
6.     int textWidth = fontMetrics.stringWidth("Season: " + Organisms.season); // Getting the
length of the screen
7.     // The x position is set to the text width - a small additional default offset
8.     // (so it doesn't end directly at the end of the window)
9.     int fpsTextX = getWidth() - textWidth - 10;
10.    int fpsTextY = 30; // Offset from the top of the screen
11.    g.drawString("Season: " + Organisms.season, fpsTextX, fpsTextY); // Draw the string
12.
13. }

```

Java

The image below represents how this appears to the user on screen.



On consideration of this, and using a mix of colour to display the text, it was noted that it is not always easily readable to the user, especially if there is a dark colour, such as deep water, behind the text. It could be argued from the image above that the first section has a high contrast and therefore may be hard to read for some. Considering the usability features aimed to make EvSim available for all users (including colour-blindness), I am going to instead add the season onto the side menu alongside the year.



This example is miles better for readability compared to the previous iteration, and also saves on needless calculations, as the added code here is much simpler, as shown below.

```

1.     // Displays the years held in GameWindow and the season held in Organisms
2.     currentYearLabel.setText("Current Year: " + GameWindow.years + " - Season: " +
Organisms.season);

```

Java

3.10.3 Testing resultant temperature change

For this test, I am going to take one example organism, and check its current temperature against the terrain attributes and the current season.

Figure 3.10.3a below shows the stats of organism 17. The season is Autumn, and it currently resides on the Beach, its preferred terrain. The attributes for a beach terrain as outlined in 3.3.5.1 define that the beach temperature should be between 13 and 23. The autumn variation is between 3 and 8 degrees increase. This means that the current temperature should be between the range of 16 and 31. The organism shown below (organism 17) has a current temperature of 19, placing this perfectly in the ideal bracket.



Figure 3.10.3 a

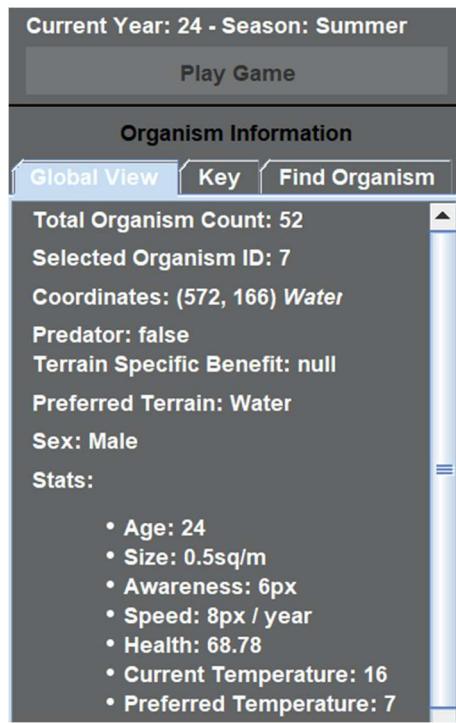


Figure 3.10.3 b

Conducting a second test on the organism from Figure 3.10.3b above, we can further evaluate the success of this section. Organism 7 is currently residing in water on the 24th year where the season is Summer. Once again, comparing the temperature defined in 3.3.5.1, the water biome should have a temperature between 6 and 13. Considering the temperature increase for the summer season is between 8 and 10, the expected range of values for the forest is between 14 and 23, meaning the current temperature of 16 is accurate.

Adding in a seasonal temperature change adds another value for organisms to counter, with their preferred habitats potentially changing as a result of this. Despite it being a small change, it is another value which adds to evolution and the cycle of organisms, increasing the scope of EvSim.

3.11 Packaging EvSim and getting user feedback

Prior to EvSim, I had no experience with exporting a Java project to an executable JAR file. After research I found that adding it as an artifact in IntelliJ allowed for the project to be packaged into a JAR file. After packaging EvSim into EvSim.jar, I proceeded to test it to ensure no errors between IDE and Java's virtual machine.

3.11.1 Testing the packaged project

An immediate issue was how the project had been packaged. Within the IDE, all files etc have been indexed into folders, meaning they were accessed through strings such as "./organisms.json". However, within the JAR file, all files are in one place, meaning they could not be accessed in the same way the IDE has. The following console result shows this.

```
PS C:\Users\Harry\IdeaProjects\EvSim\out\artifacts\EvSim_jar> java -jar EvSim.jar
Exception in thread "main" java.lang.RuntimeException: java.io.FileNotFoundException: options.json (The system cannot find t
he file specified)
    at OpeningMenu.<init>(OpeningMenu.java:36)
    at Main.main(Main.java:3)
Caused by: java.io.FileNotFoundException: options.json (The system cannot find the file specified)
    at java.base/java.io.FileInputStream.open0(Native Method)
    at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
    at com.fasterxml.jackson.core.JsonFactory.createParser(JsonFactory.java:1029)
    at com.fasterxml.jackson.databind.ObjectMapper.readTree(ObjectMapper.java:3109)
    at OpeningMenu.<init>(OpeningMenu.java:32)
    ... 1 more
```

This meant that upon execution the project did not load and the user would be presented with a useless file. To counter this, a class loader was implemented to the areas where files could not be read, as these search the entire class for the presence of the intended file, before setting it as a path which can be read by Jackson. An example of this within the project is shown below.

```
1.     try (InputStream inputStream =
OpeningMenu.class.getClassLoader().getResourceAsStream("options.json")){
2.         JsonNode data;
3.         if (inputStream == null) {
4.             data = objectMapper.readTree(new File("options.json"));
5.         } else data = objectMapper.readTree(inputStream); // Reads the JSON file as a Node
(does not require a class)
6.         windowHeight = data.get("WindowWidth").asInt(); // Gets the value stored at the key,
and returns as Integer (default is JsonNode)
7.         windowHeight = data.get("WindowHeight").asInt();
8.     } catch (IOException e) {
9.         throw new RuntimeException(e);
10.    }
```

Java

Packaging this again with the necessary changes made, the project now loads as intended, with all features being available for the user, as intended.

3.11.2 Stakeholder feedback

Before the development of EvSim can be considered concluded, I have consulted the stakeholder with the current state of the program, to ask them for any feedback or suggestions which can be made. This is beneficial as while I know what the program does, if the stakeholder does not without prior explanation, then the program is rendered useless. The program should be easily understood by the user, requiring minimal explanation. The stakeholders comments were as follows.

“Everything worked well and as expected, features were easy to understand and work with despite no explanation. Sim was accessible enough for a novice, the customizable features gave enough freedom, especially with disabilities such as colour-blindness.”

The stakeholder did however provide some suggestions which could be added/improved.

- 1) Fix the saving feature and options menu
- 2) Add Eastings and Northings (visible longitude and latitude lines) to the map

3.11.2.1 Options Menu Bug

While the stakeholder was thankfully able to install and use EvSim on their computer, issues which were found in 3.11.1 emerged again.

3.11.2.1.1 The error reason

This time however it was a very small issue which had caused the options saved to not be read from the correct file. The corrections made during 3.11.1 failed to consider the fact that the method used will cause the read file to always be within the package itself (which is therefore default), instead of the file created externally. This should therefore be a very simple fix, as the way in which the files are read can be adapted and corrected accordingly.

3.11.2.1.2 The fix

In order to read the options file correctly, two encapsulated classes (similar to how organisms are initialised) will have to be created. The two classes to create are a Colours class (containing the options for colours) and a Options class (containing the window size and colour list). The two classes are shown below, and as shown within, the Options class also initialises the Colours class as a list.

Colours:

```

1. public class Colours {
2.     private int red; // Initialising all variables with the same name and type as stored in
JSON
3.     private int green;
4.     private int blue;
5.     private String setting;
6.     public int getRed() { // Getter and setter methods for each
7.         return red;
8.     }
9.
10.    public void setRed(int red) {
11.        this.red = red;
12.    }
13.
14.    public int getGreen() {
15.        return green;
16.    }
17.
18.    public void setGreen(int green) {
19.        this.green = green;
20.    }
21.
22.    public int getBlue() {
23.        return blue;
24.    }
25.
26.    public void setBlue(int blue) {
27.        this.blue = blue;
28.    }
29.
30.    public String getSetting() {
31.        return setting;
32.    }
33.
34.    public void setSetting(String setting) {
35.        this.setting = setting;
36.    }
37. }

```

Java

Options:

```

1. import java.util.List;
2. public class Options {
3.     private int windowHeight; // Initialising all variables with the same name and type as
stored in JSON
4.     private int windowWidth;
5.     private List<Colours> colours; // The colours as a list of the colours class
6.
7.     public int getWindowHeight() { // Getter and setter methods for each
8.         return windowHeight;
9.     }
10.
11.    public void setWindowHeight(int windowHeight) {
12.        this.windowHeight = windowHeight;
13.    }
14.
15.    public int getWindowWidth() {
16.        return windowWidth;
17.    }
18.
19.    public void setWindowWidth(int windowWidth) {
20.        this.windowWidth = windowWidth;
21.    }
22.
23.    public List<Colours> getColours() {
24.        return colours;
25.    }
26.
27.    public void setColours(List<Colours> colours) {
28.        this.colours = colours;

```

```
29.    }
30. }
```

Java

Now that these classes are created, the JSON can be read through value instead of tree, meaning it is passed into a class type instead of JsonNode. This method has been used throughout the rest of the program for organism information, so placing this section in line with that also helps with consistency.

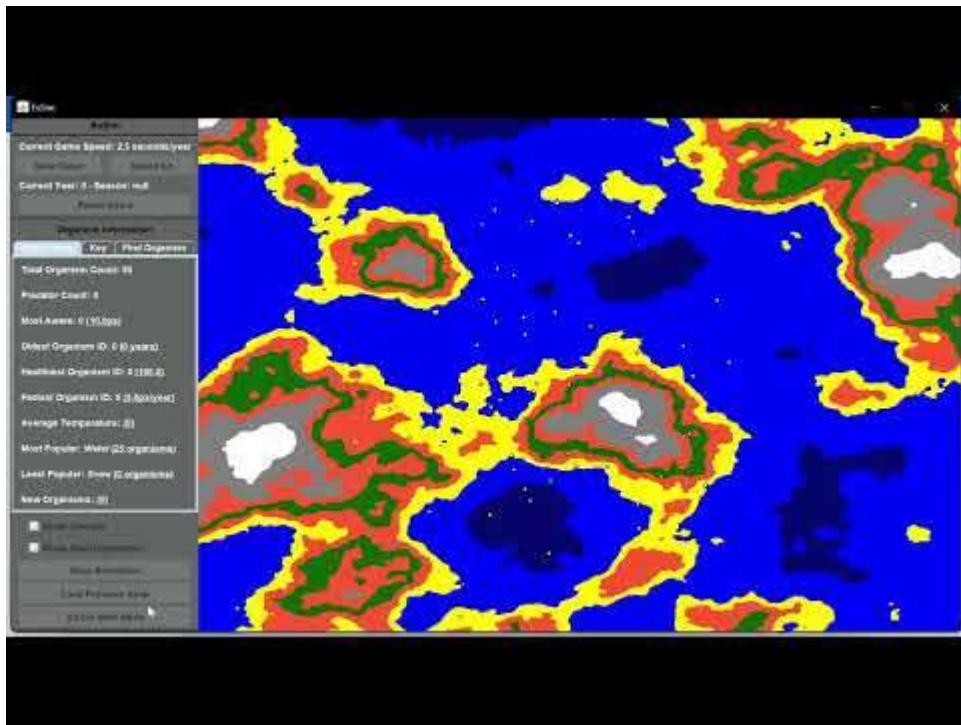
The current settings are now read and updated through the following method. The original method is shown in 3.1.4.

```
1. private void updateCurrentSettings(){ // Ensures the current settings are displayed as the
   current settings on the options menu
2.     ObjectMapper objectMapper = new ObjectMapper();
3.     try {
4.         Options data = objectMapper.readValue(new File("options.json"), Options.class);
5.         List<Colours> colours = data.getColours();
6.
7.         windowSize.setSelectedItem(data.getWindowWidth() + " x " + data.getHeight());
// The current set window size
8.         // Represented as "WindowWidth x WindowHeight", e.g. 1920 x 1080.
9.
10.        grassColour.setSelectedItem(colours.get(0).getSetting()); // Displaying the current
selected setting in the options menu
11.        waterColour.setSelectedItem(colours.get(1).getSetting()); // Takes the object at
the first index, then takes setting value
12.        mountainColour.setSelectedItem(colours.get(2).getSetting()); // get(2) returns the
object at array index 2
13.        beachColour.setSelectedItem(colours.get(3).getSetting());
14.        forestColour.setSelectedItem(colours.get(4).getSetting()); // Returned value is
displayed as the selected item on the menu
15.        deepWaterColour.setSelectedItem(colours.get(5).getSetting());
16.    } catch (Exception exception) {
17.        throw new RuntimeException(exception);
18.    }
19.    if(maxWindowHeight < 1080 || maxWindowWidth < 1920) windowSize.removeItem("1920 x
1080");
20.    if (maxWindowHeight < 2160 || maxWindowWidth < 3840) windowSize.removeItem("3840 x
2160");
21.    if (maxWindowHeight < 1440 || maxWindowWidth < 2560) windowSize.removeItem("2560 x
1440");
22.    if (maxWindowHeight < 768 || maxWindowWidth < 1366) windowSize.removeItem("1366 x
768");
23.    if (maxWindowHeight < 900 || maxWindowWidth < 1440) windowSize.removeItem("1440 x
900");
24.    if (maxWindowHeight < 720 || maxWindowWidth < 1280) windowSize.removeItem("1280 x
720");
25.    if (maxWindowHeight < 1024 || maxWindowWidth < 1280) windowSize.removeItem("1280 x
1024");
26. }
```

Java

3.11.2.1.3 Testing the fix

If the options menu now displays the changed setting, then this bug has been fixed. The video below shows both the original bug, and the new fix. As shown within, the first test showed that while the map itself loaded with the user's saved options, the menu itself did not display these changes, elongating the effort that the user has to go through each time if they want to change any settings. By fixing this, hassle is avoided and the usability significantly increases. This error was not apparent to me prior to this due to it not being a primary focus of any tests conducted during the packaging of EvSim. In the video, two EvSim states are shown, the first 20 seconds shows the old package with the error, and anything after that shows the new iteration, where changes save to the menu.



3.11.2.2 Eastings and Northings

This feature would allow the user to better understand the cartesian coordinate system which is used within EvSim. This was pointed out as the stakeholder was unable to identify where the origin of the coordinate system is. Considering how traditional cartesian coordinate grids have their origin at the middle, this is what the user assumed it to be, however the origin of the grid is actually at the top left of the map. This is how the suggestion of lines of latitude and longitude was formed. To implement this into EvSim, wouldn't be difficult, however it would be very time consuming. Lines would have to be drawn at each marker (e.g., every 100), alongside a label for them also. This would not only take up quite a large section of the screen (which could display evolution) but also may not even improve the usability in any way.

While the suggestion of Eastings and Northings in EvSim is arguably not a bad idea on the surface, the implementation would be flawed, and would serve a purpose which already has little to no impact on Evolution itself, or information which the user could use. Considering the coordinates were added to the UI for testing, and kept in as an additional feature, to spend time on adding in this feature would at most be a future revision for when maps are significantly larger or have greater detail. It could be argued that removing the coordinates from the UI would be a better option. Considering the coordinates only appear for a selected organism, which is painted in a light cyan anyways, the user does not gain or lose anything with or without the coordinates shown. Even if the user was to need the coordinates of the organism, the grid can be quickly figured out, with the organisms closer to the top left showing smaller coordinates, and the ones further away showing larger.

Despite counting this suggestion as a future improvement, I will cover how I plan to add the feature in the future.

3.11.2.2.1 Future implementation and logic

To implement lines of longitude and latitude, the approach does not have to be complex. A vertical/horizontal line would have to be added to the screen every 100 points. These lines would also have to have a label displaying the coordinate, for example 100 or 500. This would directly mirror what is found on many physical maps in real life. An example of what the overlay would look like to the user is shown below, with the grid layout and labels displayed.

0	100	200	300	400	500	600	700	800	900
100									
200									
300									
400									
500									
600									
700									
800									
900									

Implementation wise, the game window class would have to check for the feature to be enabled (similar to how console and dead organisms are current selected), before adding in the lines. The line itself would have to be added using an external library, or a rectangle flattened to appear as a line. The dimensions of the rectangle drawn would have to equal the size of the window, and only be drawn on the map itself. The lines could be added to a separate JPanel, which is then added onto the map panel. Once the lines have been drawn, the coordinates themselves would also have to be added, likely through trial and error to ensure that the labels are in the correct places.

Once these have been added in, the colours may have to be adapted or even outlined so that they do not blend too much with the background (a mistake which became apparent in 3.10). This is once again a lot of trial and error involved to find a colour which does not only contrast, but stand out.

Considering the amount of calculations, trial and error, and sheer overloading of code which would be required to add this, for a feature which serves little purpose in the larger picture, it is clear why I have chosen to not include this feature in the current version of EvSim. It is however an option to include this feature once the map and other UI features have been improved.

4.0 Post development testing

This section brings together the testing of the whole program, testing both function and usability. Due to the complex nature of EvSim, I am going to record a full simulation of the aspects of each, with commentary explaining the elements. The tests will look at how EvSim handles processing, how it links to the backend, and how the user can use this information. The usability test will look at how the program interacts with the user and their needs/mistakes. For example, the colourblind features, how they can be used, and the benefits of using it, but also the handling of exceptions when the user enters an unexpected value. Each of these tests analyse EvSim's ability to work with the user and the overall success of the project.

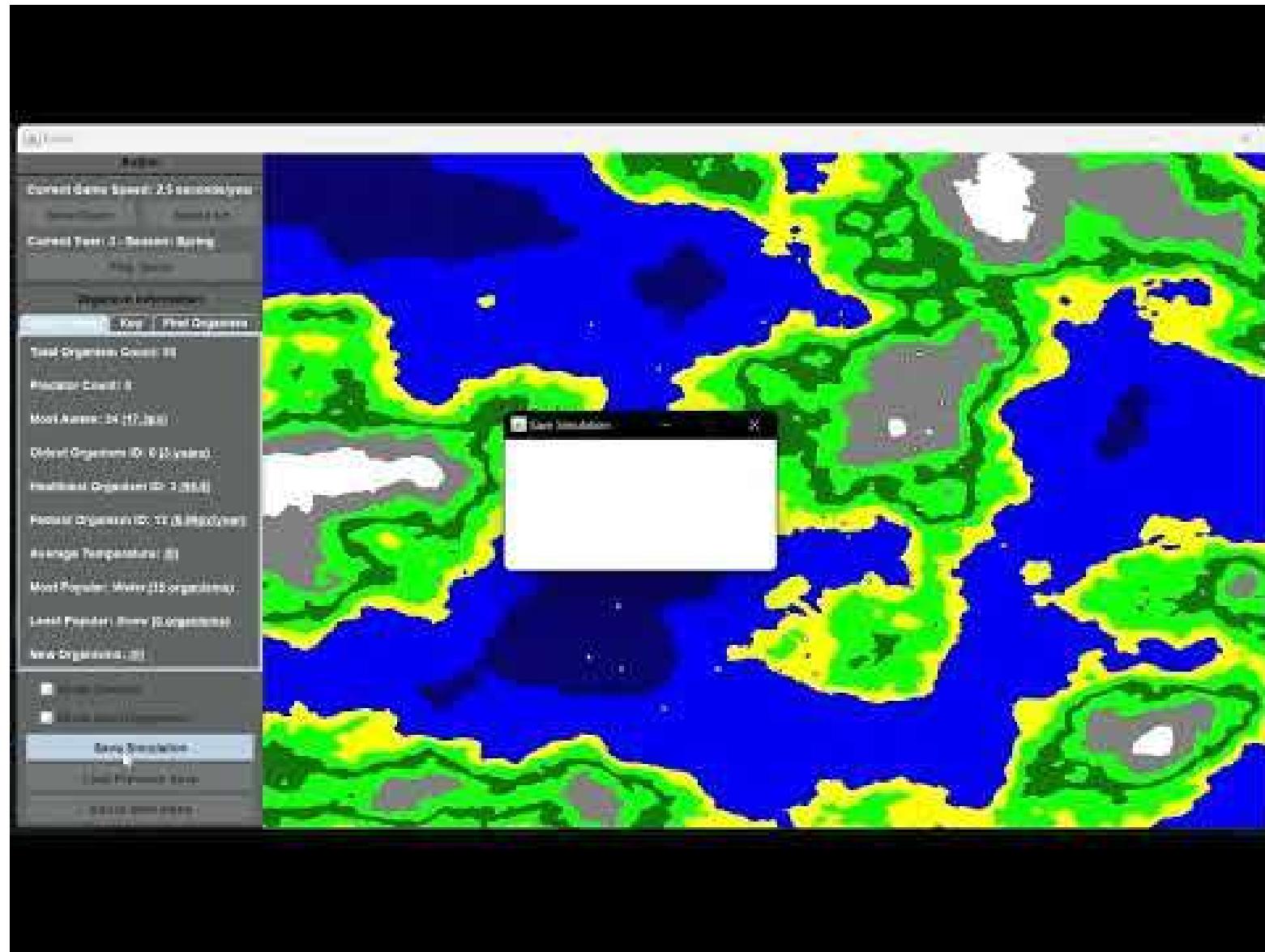
4.1 Testing for function

As mentioned, this section covers how the program processes and interacts with both UI and backend. I will cover tests for the physical attributes of EvSim, such as the evolution cycle, and the process of loading and saving. The tests performed are outlined below.

Test #	Description	Test Data	Expected Outcome	Actual Outcome	Evidence	Timestamp
1	The user should be able to define the number of organisms to display before starting the simulation.	5 Organisms	5 Organisms should be shown and displayed to the user once the simulation has started.	As expected, see commentary	Video 4.1a	0:00
2	The user should be able to define the map seed and size before starting the simulation.	Map seed of 192. Screen size of 850 x 700	The map shown in a window size of 850 x 700.	As expected, see commentary	Video 4.1a	1:14
3	The chosen map seed should be created and displayed to the user, both within the map preview and the simulation itself.	Map seed of -362	The map seed of -362 should be identical throughout, in the preview of the map, in the simulation, and after a restart of the program.	As expected, see commentary	Video 4.1a	2:39
4	The map seed which is generated is tested to ensure it represents the expected data.	Map seed of 500	The map should follow the following layout in order of contact: deep water, water, beach, grass, forest, grass, mountain, snow.	As expected, see commentary	Video 4.1a	4:37

5	The save simulation function should save the current game state.	The active JSON (./organisms.json) should be compared to the stored JSON (./Save Games /test/organisms.json)	The two files are the same.	As expected, see commentary	Video 4.1a	6:18
6	The data displayed in the side menu should replicate what is stored in the active JSON file.	A new test simulation, the active JSON file (./organisms.json)	The data shown on the side menu for a selected organism should replicate what is stored in the JSON file. The awareness should be displayed as 10x the value stored.	As expected, see commentary	Video 4.1a	10:16
7	Organisms will be tested to ensure that evolution is correctly displayed.	A saved test simulation, from around the year 200. The key displayed on the side menu.	The key which displays organism appearance should match organisms within the simulation with similar attributes. For example a predator organism should appear the same as displayed in the key (red outline).	As expected, see commentary	Video 4.1a	13:34
8	Organisms become asexual in water more frequently than they do out of water. This may require testing of many simulations to find one where land and water is balanced.	New simulation.	Organisms in water should become asexual long before a land based organism does.	As expected, see commentary	Video 4.1a	17:08
9	The program operating with changing parameters. This is the visual display to the user. Any changes such as movement should be shown frequently and accurately.	A new simulation.	The simulation should update the organisms frequently over a period of 50 years. Organisms should move about the map, change in size, and	As expected, see commentary	Video 4.1a	19:23

			give birth to other organisms.			
10	The user's ability to search for an organism should be swift and accurate.	A new simulation. Random organism.	The organism ID which was searched for should return the linked organism and its attributes. The organism should be highlighted on the screen, and the attributes displayed on the side menu.	As expected, see commentary	Video 4.1a	22:32
11	This tests the family tree of an organism, checking that the attributes are similar.	A saved test simulation, from around the year 200. One organism will be chosen that is currently alive. The active JSON file (./organisms.json). A random selection of related organisms.	The family tree of the organism should follow in line with the current attributes of the organism. For example, the size of the organism should not differ far from the parents' sizes.	As expected, see commentary	Video 4.1a	24:48



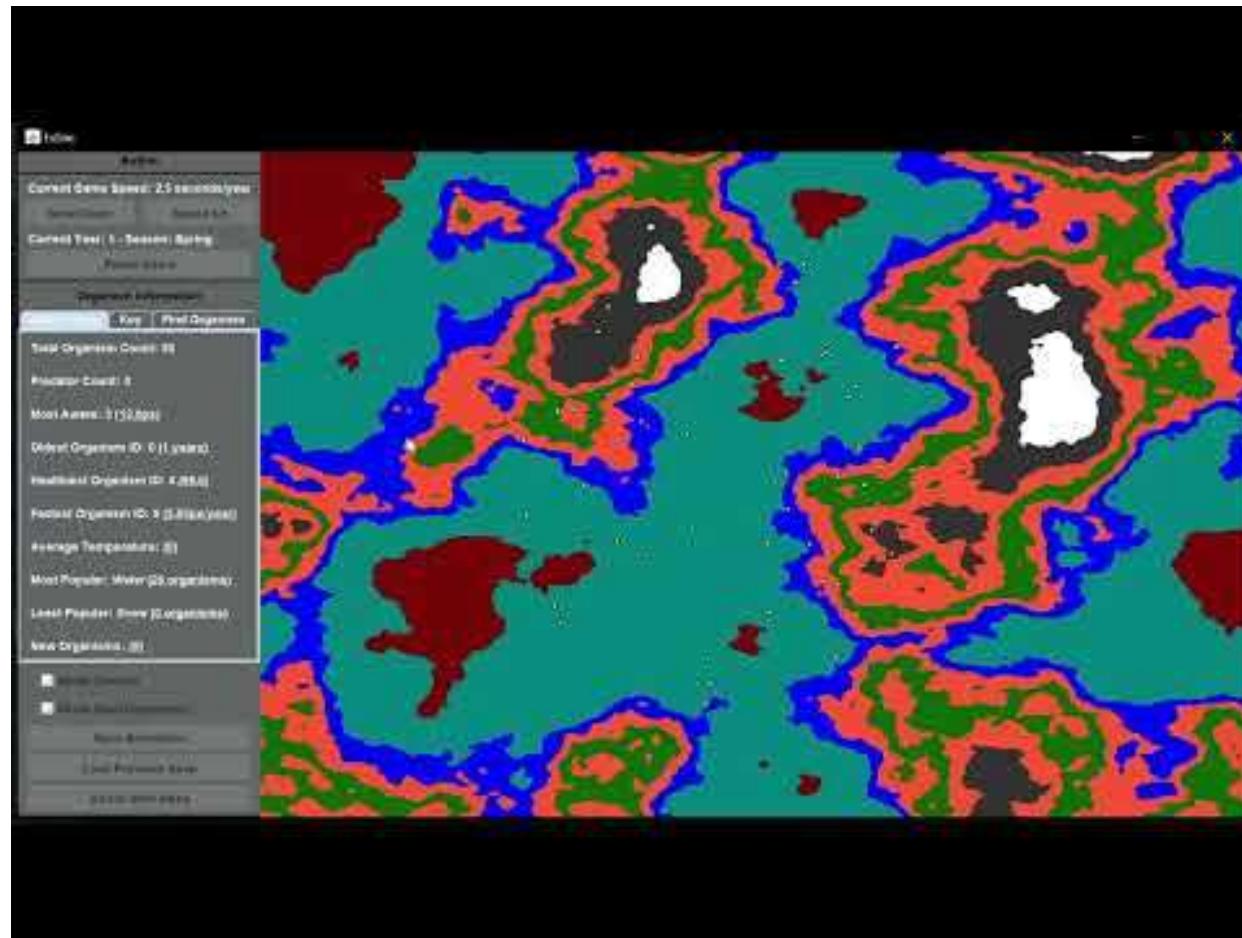
Video 4.1 a

4.2 Testing for usability

As mentioned also, this section covers how the program interacts with the user. The user's ability to change how EvSim works, for example the colour blind options will be tested to ensure that the changes the user makes both work and save for future use. Furthermore, this section will test how the program reacts to user mistakes, such as entering wrong values or strings. There is little user input for EvSim, however there are still some, which all require testing. This section will use a mixture of both commentary and screenshot evidence, all labelled accordingly in the table below.

Test #	Description	Test Data	Expected Outcome	Actual Outcome	Evidence	Timestamp
1	The main menu should lead the user into starting a new simulation, loading into an old one, change the appearance of EvSim, and quit the game.	N/A	All buttons on the main menu should function with the intentional outcome. The 'Start' button should lead the user into the selection menu. The 'Load' button should lead the user to select from a list of previous simulations. The 'Options' button should lead the user to select their colour and window size options. The 'Exit' button should allow the user to quit the program, ending its operation.	As expected, see commentary	Video 4.2a	0:00
2	The application should allow the user to customise the apps appearance. The colourblind accessibility options should display to the user the change that it will make, allow the user to save that combination, and make the change have affect.	N/A	Any changes made to specific terrains should be previewed on the menu, with any changes saved being kept within the program, even after restarting.	As expected, see commentary	Video 4.2a	1:21
3	The map should be painted in the same layout as the users selected colour scheme.	N/A	The colours selected and saved from test 2 should be replicated on the actual map which is displayed to the user once the simulation is started.	As expected, see commentary	Video 4.2a	3:11
4	The side menu should allow the user to speed up, slow down, and pause the simulation.	N/A	The speed up, slow down, and pause game buttons should carry out their respective effect on the simulation.	As expected, see commentary	Video 4.2a	4:09
5	The side menu should allow the user to view specific organisms, and global attributes.	N/A	The Find Organism tab should allow the user to input an organism ID and select find. If the organism is found then the side menu should	As expected, see commentary	Video 4.2a	5:30

			display the organisms statistics, and if not it should display an appropriate error message.			
6	The side menu should allow the user the option to load, save, and quit to main menu.	N/A	The save, load, and exit to menu buttons should do their respective actions with no errors.	As expected, see commentary	Video 4.2a	7:22



Video 4.2 a

5.0 Evaluation

This section judges the extent to which essential features, success criteria and usability features have been met. Each section will be evaluated to analyse to what extent I have met the criteria defined, why I have come to this conclusion, and what could have been done better. Each section will outline the judgements and the decisions made, and why they were made, as well as a justification for the changes.

5.1 Comparing success criteria and test data

This section specifically focuses on the success criteria laid out in 1.7, how it compares to the test data from 4.0, as well as the improvements that could be made to better meet this criteria.

5.1.1 Allowing the user to change and set attributes during the creation of the simulation

The first section of the success criteria outlines that the program should allow the user to change attributes of the simulation before it is loaded, such as the attributes of organisms and the map.

This section of the criteria has been met early into development, 3.1.4 specifically. With the start game menu allowing the user to define the map seed (and preview it), organism count, screen size and the ability to enable/disable seasons and predator prey cycles. Figure 5.1.1a shows that I have met this criteria.

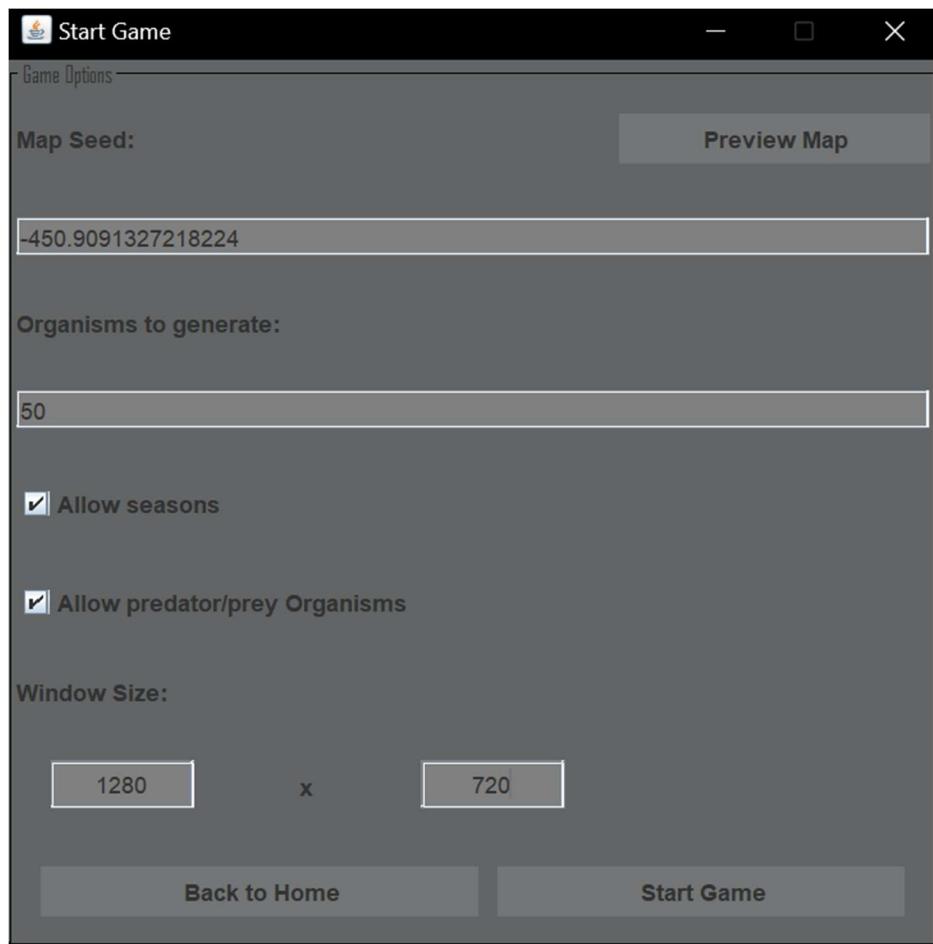


Figure 5.1.1 a

5.1.2 Manipulation and ability to save currently processing formats

The second section of the success criteria outlines that the program should be able to load and save currently processing formats, as well as the option to pause and change speed of the simulation.

This criterion is mainly fulfilled through the use of the side menu within the game. Figure 5.1.2a shows that the user has the options to speed up, slow down, pause, save, and load each simulation. All of these features were mentioned within the success criteria, and with their functionalities working as intended (see 4.1 Post Development Testing), this criteria has been met.



Figure 5.1.2 a

5.1.3 Displaying organism attributes

The third section of the success criteria outlines that the program should display actively at least 7 of the organism attributes defined in 1.4.

Once again this is contained within the side menu once the simulation plays out. The side menu displays a total of 12 types of organism statistics, such as awareness, size, and speed. Furthermore, the side menu also displays global statistics showing the highest attributes and statistics. Overall, there are 26 labels displayed to the user containing data which can be used for hypotheses. This criteria has been met.

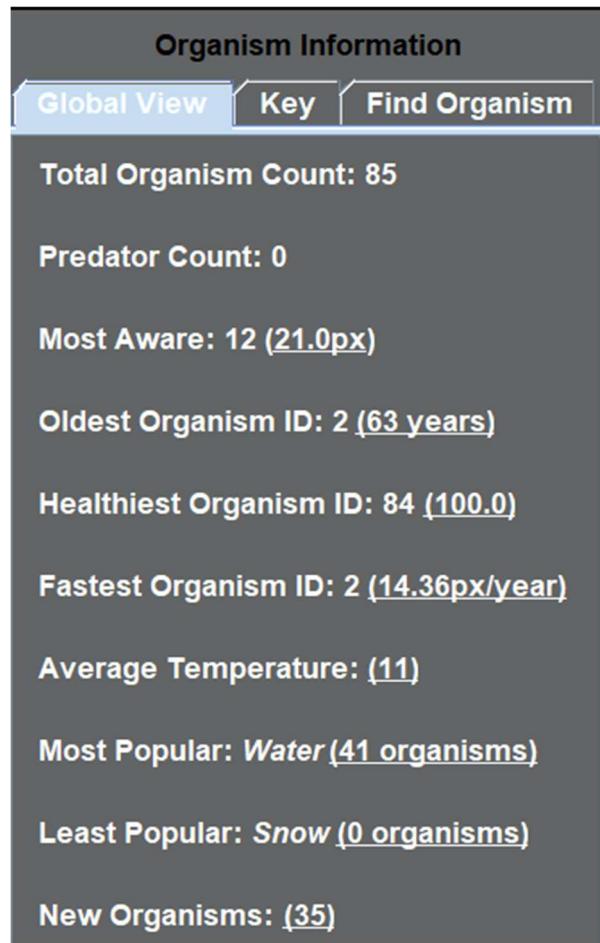


Figure 5.1.3 a

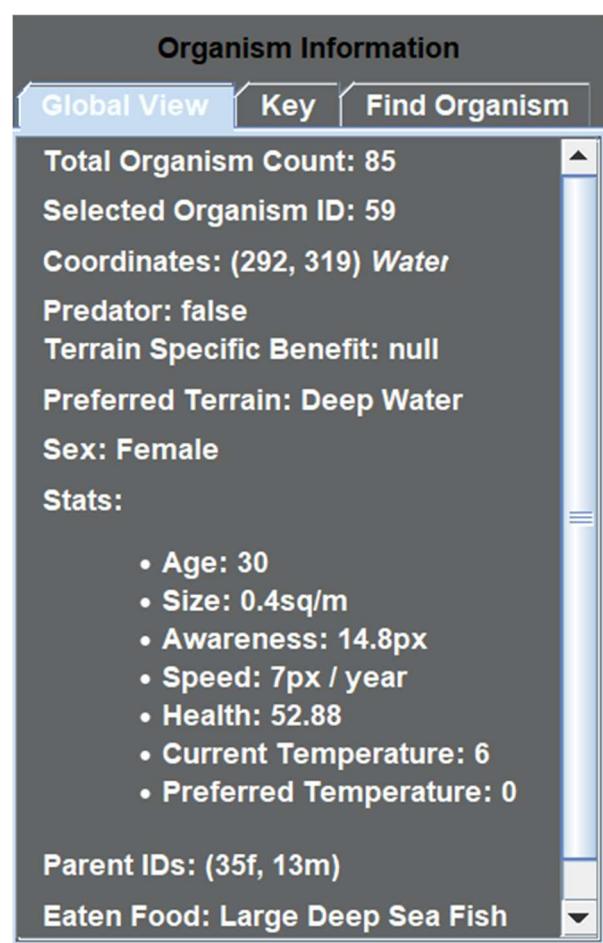


Figure 5.1.3 b

5.1.4 Creating and displaying a sufficient world which has fluid biomes

The fourth section of the success criteria outlines that the program should generate and display a fluid environment for the organisms which resembles Earth in many ways. This section also mentions that the created world should contain organisms.

During the generation of the simulation, the user can preview a map, or generate the simulation. Once the simulation is generated, the selected seed is sent and created, creating a world which resembles Earth in many ways, as noted by the stakeholder. The world contains biomes that are found in real life, with logical tropes such as higher mountains containing snow and water having deeper sections commonly found in the middle of a large body of water. The organisms are also displayed in a way in which their attributes are labelled, and shown through colour to the user.

While the generation of the map can be considered a success against the success criteria, the generation of organisms may be a little confusing to a user without prior explanation of meanings, or the key not being seen. For example, the stakeholder did not know that predator prey cycles existed until they were pointed out against the key. Ideally I would have preferred to display the organisms as more than circles, potentially through other shapes such as triangles, or even through images. However, the default JDK-17 package I am using does not include this ability without the use of complex polygons, which may have taken valuable time to create. Therefore my implementation of this was not possible without exploring external libraries, an option which was disregarded due to time constraints.

In the future, a library such as JavaFX could be used to draw more complex shapes such as triangles, due to its use of much simpler points. This therefore means that this criteria was met half way, with both the map and organism display being suitable, but not as usable as previously hoped.

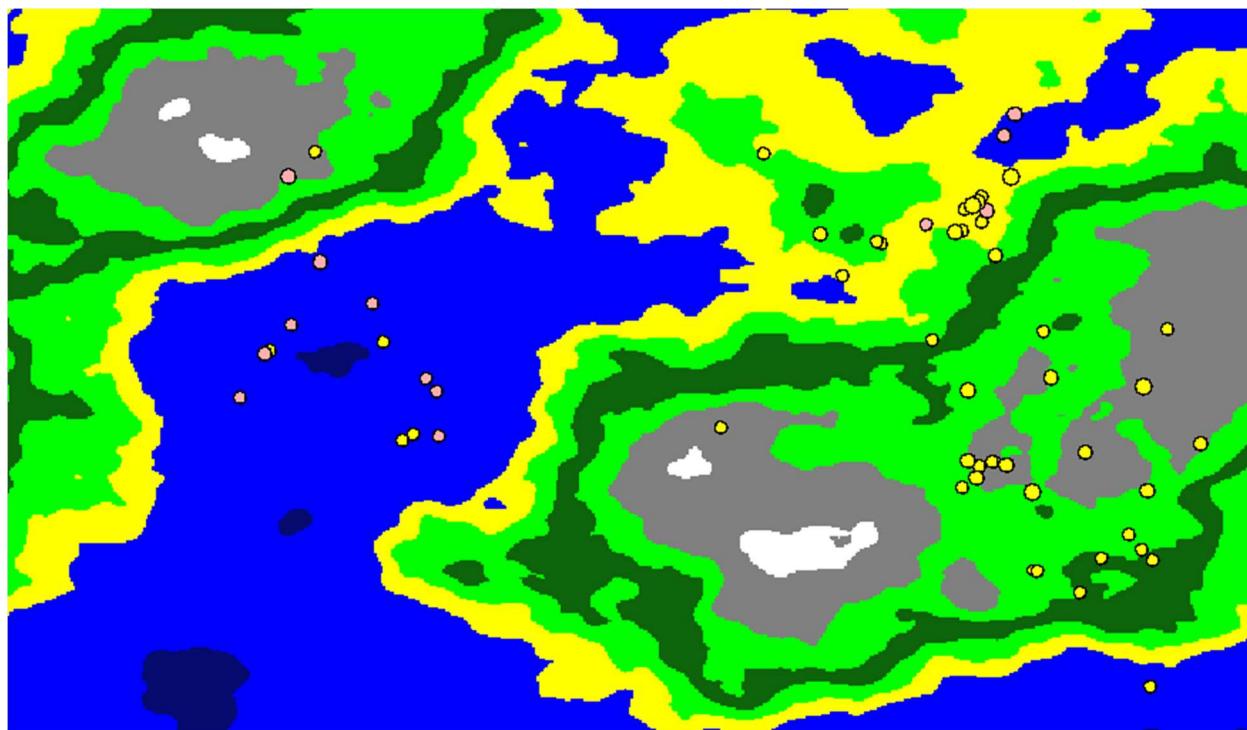


Figure 1.5.4 a

5.1.5 Demonstrating evolution physically and visually

The fifth section of the success criteria outlines that the program should demonstrate evolution visually through the organisms changing shape, and physically form their factors developing depending on their environment.

Speaking strictly on the organisms demonstrating evolution physically through their environment, EvSim demonstrates this in many ways, from adaptations such as size, speed, awareness, and food. All of these factors take contributions from their current environment in many ways, for example the awareness taking into account the general visibility of the terrain (deep water, very little visibility). While it is incredibly difficult to produce a program which mirrors the characteristics of real life, I believe that EvSim has achieved the ability to demonstrate evolution physically, through factors such as aforementioned. Attributes are fluid (interact with each other, for example awareness and speed) and mirror many characteristics of real life while not being an exact mirror. *See 5.1.3b*

Visually, as mentioned in 5.1.4, due to the JDK limitations and lack of time, it was difficult to demonstrate evolution physically. While custom images could have been used, it would still take up a lot of time which could have gone towards other areas of development. After providing the stakeholder with the program, they agreed that changing attributes are hard to track without sufficient explanation, however it was agreed that organisms react and evolve to their specific environment. This could have been met more thoroughly with another revised version of the organism movement, however with the algorithm already becoming extremely expensive to process, any further development may have hindered the program. In the future, I will take into account an organism density factor, which should prevent the organisms from ‘bunching’ together throughout much of the simulation.

Considering that after explanation to the user of how EvSim works, they are able to understand and work with it fluently, and the fact that the UI was not of key focus throughout, I have met this criteria. *See 5.1.5a*



Figure 5.1.5 a

5.1.6 Calculating and storing changing attributes

The sixth section of the success criteria defines that the program should be able to both calculate and store the attributes of the organisms efficiently enough so that it is both easy to handle and can be displayed to the user.

The calculations of EvSim are stored in a mixture of a class array, and JSON packages. The calculations which take place are updated to a globally defined array, which is mirrored to the back-end database (JSON) after each 'year'. The calculations are able to be processed for over 1000 organisms in less than half a second, showing that the program is not only efficient enough to be ran on most machines, but also flexible, due to the ability to slow down and pause simulation time.

While causing many issues throughout, getting JSON implemented in a way which was easy to work with early on in development proved to be a major help later on, and allowed for a physical state of the simulation to be saved. While there are many elements of JSON which are a hindrance, it has allowed for the store of attributes to be maintained consistently in a way which is organised and easy for the program to interact with. I have met this criteria.

5.1.7 Operating with changing parameters

The seventh section of the success criteria instructs that the program should be able to operate sufficiently with changing parameters, such as the organisms moving and adapting.

Once again, due to the fluidity and flexibility of JSON as a back end data storage, the organisms are able to be updated yearly, in a way which is not a smooth transition, but more of an insight into where the organism could have travelled during the year. Any changes which are made are not only saved to useable format within the program (array), but also to the database. This means that the changes made are saved physically and virtually, allowing them to be read and the changes made visually to the user, for example the movement of organisms and the updating of stats on the side menu. This criteria has been met. *See 4.1 / 4.2.*

5.1.8 Taking inputs from the user during execution of simulation

The eighth and of the success criteria states the program should be able to take inputs from the user proficiently, specifically for desire to change factors for specific organisms or terrains.

Unfortunately, this criteria has not been met. While it is not impossible for the user to manipulate a save file to change attributes, it is not intuitive at all and significant knowledge of JSON and EvSim's functionality is required. The main reason why this criteria was not met is the fact that UI development can take up a lot of time and space. In the future, I would integrate another UI menu which would allow for the user to change elements specific to a terrain or an organism. The only input a user can make during the simulation, is to find a specific organism. A feature which has little to do with the execution. I have not met this criterion.

5.1.9 Demonstrating evolution over ‘generations’

The final section of this criteria relates to evolution itself, and how evolution is demonstrated to the user over time.

As shown throughout the tests in 4.1, organisms take certain attributes from their parents, and certain attributes from their environment. For example, attributes such as size are taken from the parents, whereas asexuality and predation are environmentally based. The figures below show a few examples of what the simulation looks like after extended periods of running. There is plentiful variation in the evolution which can take place, for example in some cases many organisms are large in size, where in other cases they are small. There is also a mix between sexes and evolutionary adaptations, for example commonality of predation. From testing, a simulation does never appear to run out, running continuously. This criteria has been met.

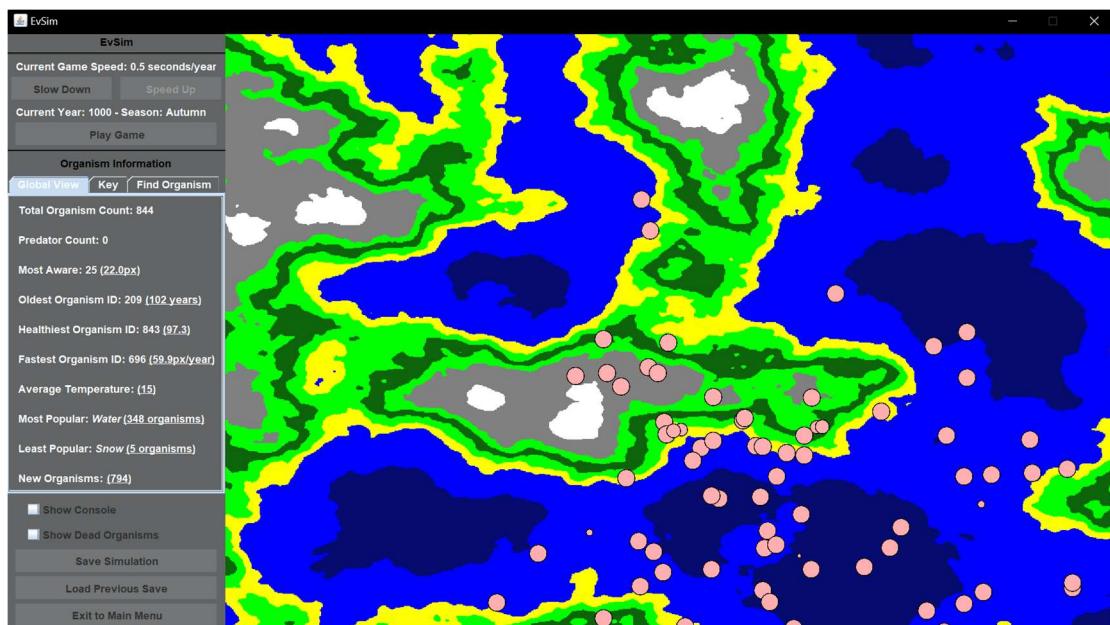


Figure 5.1.9 a

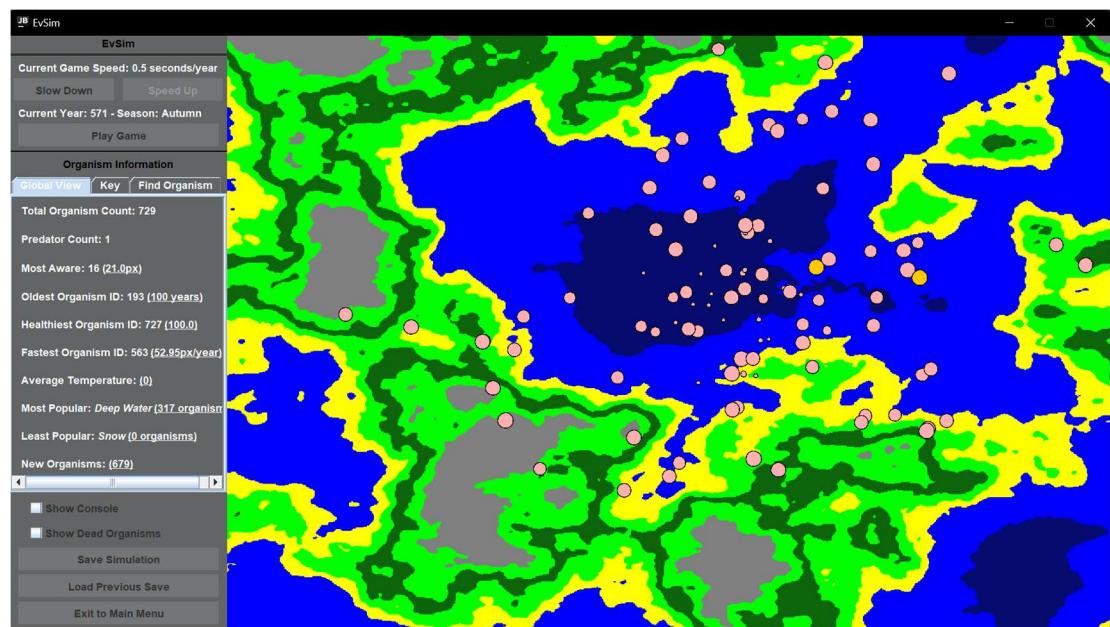


Figure 5.1.9 b

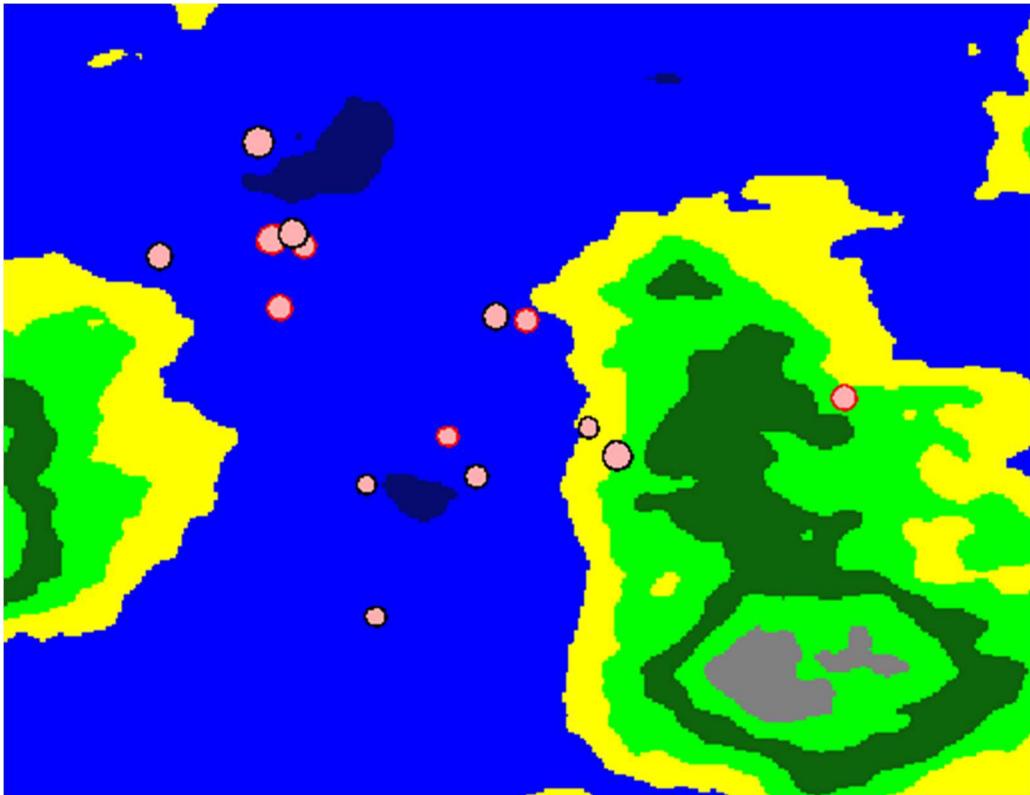


Figure 5.1.9 c

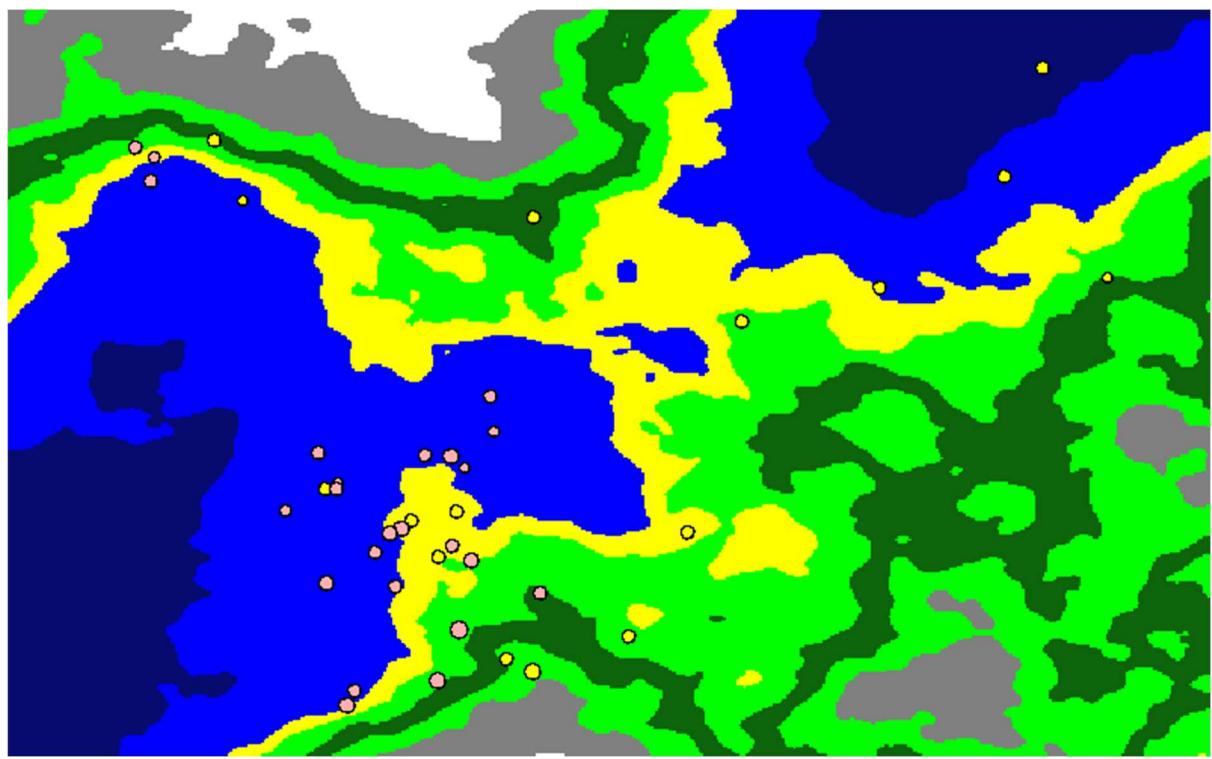


Figure 5.1.9 d

Overall, I have fully completed 7 of the criteria, with one half completed. In many cases, the reason for lack of completion is a result of the resources that were available, for example the display of organisms. While the eighth criteria was not met, adding it would have taken time which was used for balancing evolution, arguably the key aim of EvSim. I will definitely add the option for the user to change attributes in the future, as it is only using features currently available/seen previously, such as a UI form.

5.2 Assessing Usability

Many of the features mentioned in 2.4 usability features should be evaluated for their presence and success in EvSim.

5.2.1 Opening menu

This feature in 2.4 was planned to be laid out as a burger style menu with options such as Start, Load, Options and Quit. This area of accessibility was justified in the fact that it is a common trope of many applications and websites. In the final version of EvSim, the opening menu has achieved this, as seen in figure 5.2.1a. The left hand side is laid with 4 buttons, labelled exactly as designed. Each of these also have a tooltip, allowing the user to understand what the button does and its purpose without having to click it. Speaking of other general menus and UI features within EvSim and the opening menu, a similar layout is followed, with buttons containing tooltips and accessible names, as shown in figure 5.2.1b. The only improvement I could make to these menus is potentially the colour. The main colour scheme for menus in EvSim is a grey and black colour, which in some cases may not be the best choice of colours. However, this is a very small issue and the stakeholders have had no issue using the menus as intended.



Figure 5.2.1 a

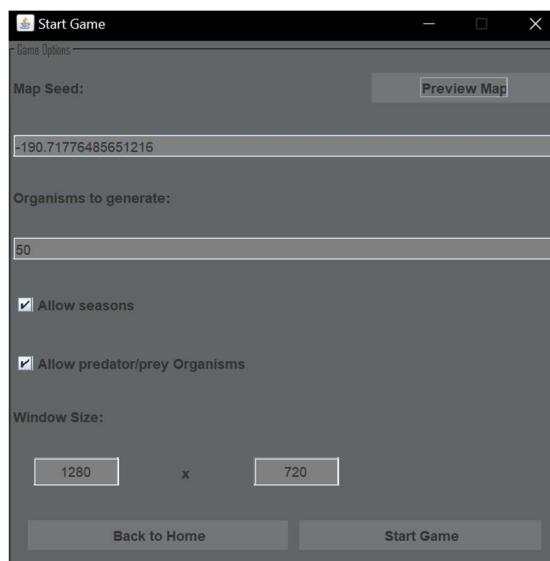


Figure 5.2.1 b

Due to many of the features seen here being common across many accessible and intuitive applications, the first target of usability is achieved.

5.2.2 Colourblind accessibility

This section focuses specifically on the display of the map, due to it being colour based, the usability features designed intended to allow the user to change the map colours to something which they can interact and understand. The options menu (accessible from the opening menu) allow for the user to select the colour of each terrain (apart from snow which is white). Once an option is selected, the user is also given a preview of what these colours look like. This means that the user can mix and match the colours to what is best for them if the preset selections are not perfect. Allowing the user to change the terrain colours like this adequately meets the target for colourblind accessibility.

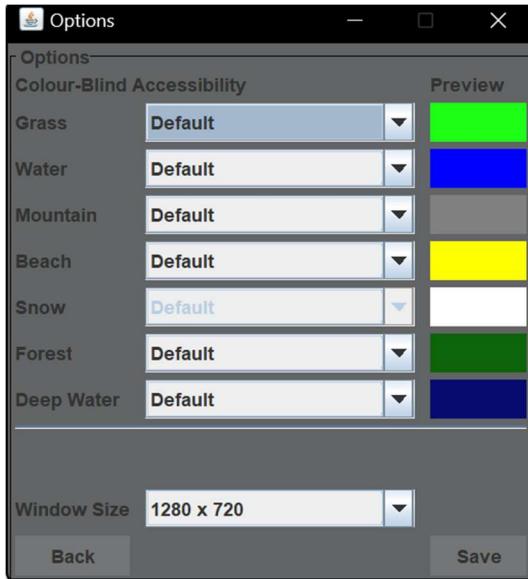


Figure 5.2.2 a

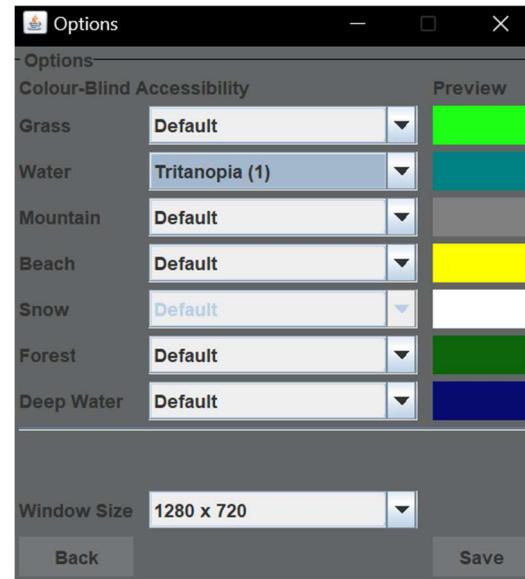


Figure 5.2.2 b

Figures 5.2.2a and 5.2.2b demonstrate how easy it is for the user to edit the colours, with them being labelled appropriately (for example tritanopia). These changes are also hard saved, meaning the user does not have to change this every time the application is loaded (see 4.2). The result of figure 5.2.2b is shown below in figure 5.2.2c.

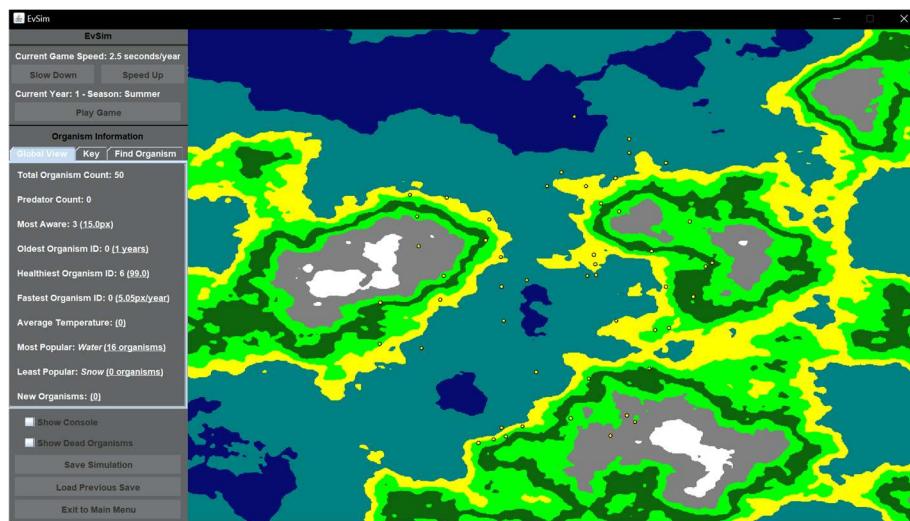


Figure 5.2.2 c

Considering that the user has the option to easily modify the display of the map, this section's target for accessibility has been met. While keybinds were mentioned in 2.4, there are no keymaps currently in place for EvSim, potentially an area which could be improved upon in the future, for example the space bar to pause the simulation, or + and – to speed up and slow down the simulation.

5.2.3 Side menu

The side menu was intended to allow the user to have more access to information, while also allowing them to change certain parameters about the application. Whilst the criteria for changing organism and terrain attributes was not met, the user can still interact with the simulation itself. Figure 5.2.3a shows that the user has the options to speed up, slow down and pause the current simulation, as well as loading and saving into a new simulation. These were all mentioned within 2.4 usability features. Other features which were mentioned is the ability to view not only global averages, but specific organism attributes once they are clicked. As demonstrated and tested in 4.2, the functionality of these work as intended. Considering these factors, and the limitations of not achieving editing organism attributes, this area of accessibility has been achieved.



Figure 5.2.3 a

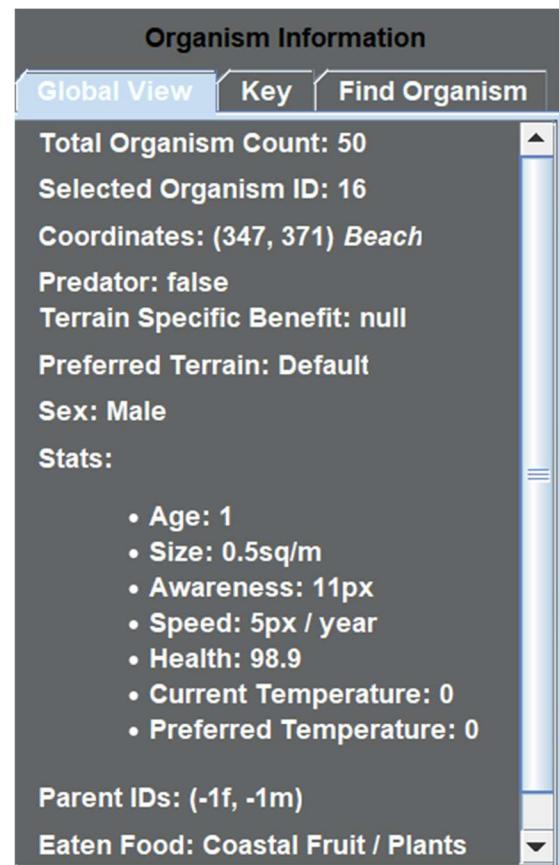


Figure 5.2.3 b

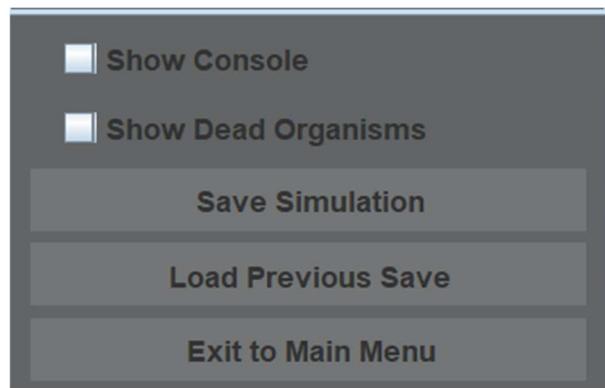


Figure 5.2.3 c



Figure 5.2.3 d

5.3 Maintenance and Limitations

5.3.1 Limitations

v5.3.1.1 Visual organism adaptations

As mentioned previously, the biggest limitation I faced throughout the development of EvSim was the inability to represent organism adaptations visually. As a factor of the success criteria, this was quite a major section of focus towards the end of development.

Ideally, the program should have been able to represent evolution through more recognisable shapes, such as hexagons, triangles, squares, etc, or even through images. However, the only options available to use in JDK-17 by default are circles and rectangles. If I was to go down the route of alternating shapes, an external library would need to be implemented, one such as JavaFX which allows for shapes to be drawn through points. The main reason why this feature was not implemented was partly time and partly complexity. Due to the majority of development time being taken up by evolution itself, by the time it came around to finishing the front end UI, the ability to find and implement a new library would have been rushed, potentially leading to a worse outcome.

While this is listed as a limitation, it is not completely limited in the fact that organisms are still represented uniquely through the use of colour. While these colours are not immediately intuitive to the user, through use of the key and time spent with the program, the user is able to get a quick grasp of what is being shown. It would definitely be a future improvement to make, perhaps even the first improvement I would make.

5.3.1.2 System resources

Throughout the development of EvSim, I alternated between the use of a laptop and a desktop computer to manage workload. While alternating between these, it became apparent that EvSim could be quite demanding on system resources. Whilst my laptop is not statistically one known for poor performance, the differences between its compilation of EvSim compared to that of the desktop was definitely notable.

While I have not tested EvSim on a hardware limited computer at this stage, it may become apparent on a slower device that the performance lacks. This is entirely as a result of the sheer amount of calculations that the program has to execute at one time (movement – every organism checking a radius, evolution – every organism updating every attribute). While there are factors within EvSim allowing for it to process in sufficient time, such as the use of time (with 0.5 seconds as a minimum), these factors do not prevent issues completely. Throughout development I have made undocumented approaches to prevent unnecessary use of system resources, such as the organism not being updated in any way when dead, and certain stats only checked if required. Furthermore, if the program is not able to execute the year fully in the time given, the time is increased by half a second to allow more time for execution during the next year. Implementations like this protect the end user from stutters (organisms not appearing on screen), and also allows the program to automatically catch up in the event execution is slowed.

While this limitation is not a significant disadvantage to EvSim, it is important to mention it due to its potential significance. An improvement I will make to EvSim in the future is simplification of certain factors and calculations, potentially even the removal of certain complexities which do not add to the representation of evolution. It is also important to mention that for the most part of EvSim's execution, it runs smoothly and without problems, it is only when the organism count reaches levels above 1000 it begins to struggle, for obvious reasons. In this event, it could be considered that a prune of organisms could take place, for example the removal of first, second and third generation organisms which no longer have an effect on the simulation itself.

5.3.2 Maintenance

This section speaks strictly on the maintainability and serviceability of EvSim, more specifically the program's structure and code. While I attempted to stick as close as possible to the structures laid out in design (see 2.1 and 2.2), this was simply not possible due to the added difficulties and unexpected requirements that surfaced throughout development. The final structure of EvSim is as follows.

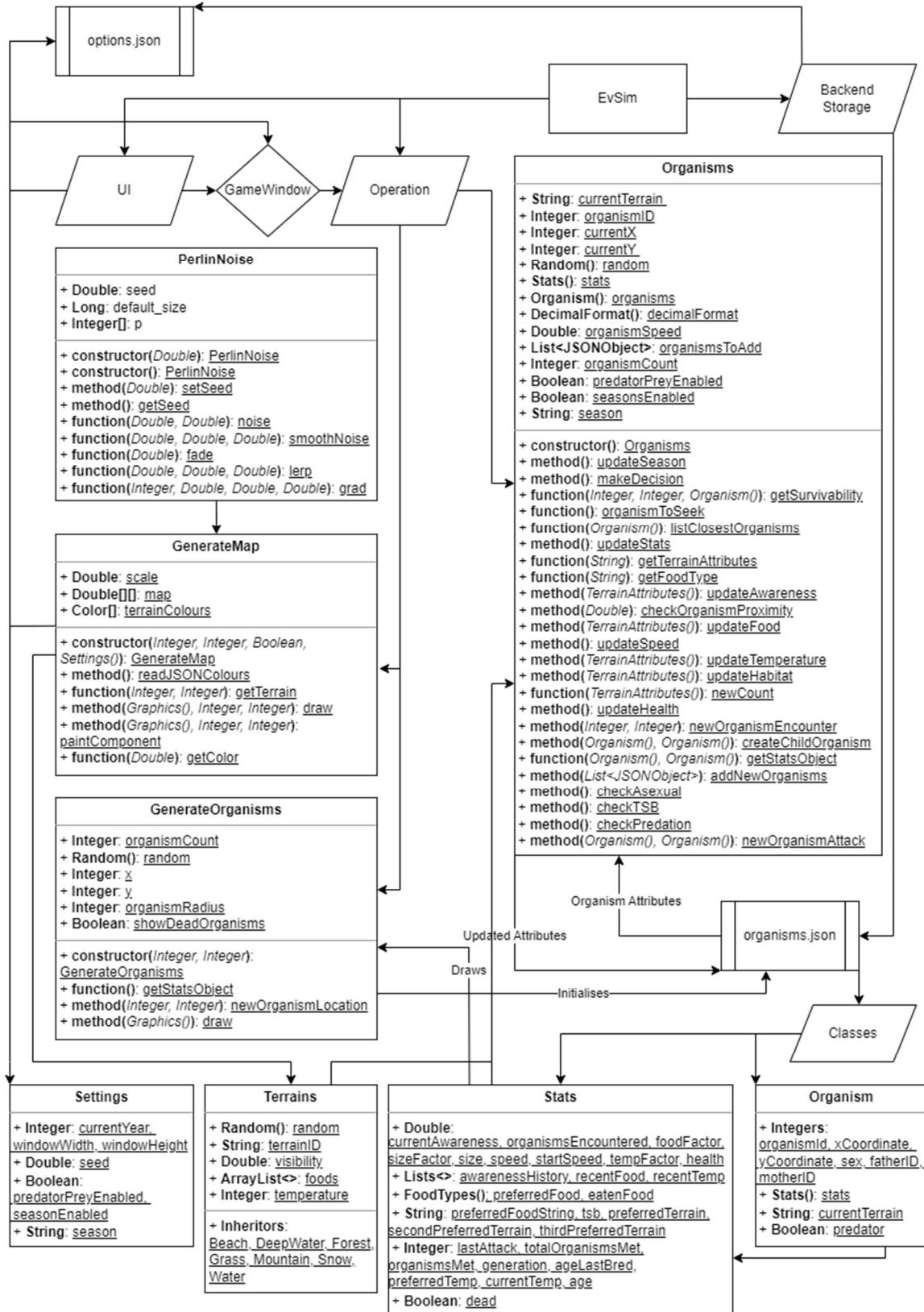


Figure 5.3.2.1 a

On first look, the class structure of EvSim may appear complex, with many of the classes interacting with each other. However, with many logical naming systems and comments within, the structure can be easily learnt. The classes are few and modularised, most in grouped in common chunks (such as the generation of the map and organisms, and the organisms being updated). The largest class, Organisms, controls how organism evolve, and move, explaining its size. This is the longest class of EvSim, but with conventional naming systems within and comments explaining sections, it is not impossible to maintain.

Many of the sections are split into modules, or encapsulated classes which can be instantiated and used across the program. Take the Organism class for example, an encapsulated class which links the Stats class, each being able to contain the contents of the back-end data base. These classes are initialised with the JSON data at the start of the year, updated throughout, with the data then being saved back into the JSON as a physical store, before the process repeats. On the other hand, there are also classes such as Terrains and FoodTypes (omitted from class diagram due to size, see 3.3.2 for class diagram of FoodTypes) which contain the relevant information for both food types and terrains through inheritance. The terrain data is stored in individual classes, and accessed when needed through the terrains package. Sorting large sections of similar data into packages much like what has been done here further reduces the complexity and overpopulation of classes, as they can be accessed separately.

Unlike languages such as JavaScript and Python, the type safety of Java is incredibly robust. Variables cannot be defined without an explicit type, such as integer or string. This means that there are little to no type errors or fault within the program. Forward planning of attribute information, specifically in the Stats class means that variables are defined logically as needed, for example an integer representing an organisms age and a double representing the health.

Due to this project demonstrating evolution, a situation which could be interpreted and demonstrated in many different ways, the size and complexity of the program is justifiable.

5.4 Future Improvements

EvSim is definitely not at its finished stage, and I definitely would like to continue with its development, as I have enjoyed researching and working on EvSim in my free time. Technically, with enough optimisation, there are endless possibilities with EvSim, due to its robust design allowing additional implementations, a factor which is in many ways a result of JSON as the choice of a back-end database. In this section however, the main future improvements which I would like to make are defined and explained.

5.4.1 Organism movement

In total, the organism movement was revised 4 times throughout development. Additional attributes being added and factors that required additional attention such as predation and asexuality meant that the movement was constantly updated to include these factors. Despite these various attempts at creating an efficient method to move organisms, it is still not ‘perfect’. The main issue present at the moment is that organisms only seek out other organisms, and do not run away. This means that organisms tend to bunch together, especially during the latter stages of the simulation. This was also noted by the stakeholder during their test (see 3.11.1). While the movement is calculated, with many aspects taken into account, it can definitely be improved, for example introduction of an organism density to deter organisms away from densely populated areas, which would remove the chance for organisms to bunch together.

5.4.2 Changing attributes during execution

This change is an obvious one, due to it being one which I was unable to complete before the end of development, and the only factor I did not achieve of the success criteria. The implementation of this shouldn't be too difficult, the most difficult thing would be to link the front end and back end. However, with the amount of information to fetch and update, as well as the validation for each input would be a lengthy process requiring a lot of testing. The UI would have to display to the user information in a similar way to the side menu and start menu does currently, except there are many more values to display and change. As mentioned previously, this addition would bring much more usability to the user as they have more choice of how the simulation plays out, potentially allowing for more sophisticated hypotheses to be carried out.

5.4.3 Improving the appearance

5.4.3.1 Map appearance

Improving the appearance of the map is something which I have looked into since the very start of development. The implementation of Perlin noise as a map generation tool simplified the creation of the map quite significantly, and opened the door to many options. The appearance of the map during the first implementation (see 3.1.1) was not what I had in mind, and whilst the implementation of Perlin noise (see 3.1.1.1) was adequate for the program, there was always the potential to increase the appearance of the generated map. During research which was carried out to find Perlin noise, some of the examples I found were quite detailed, see 5.4.3a for example.

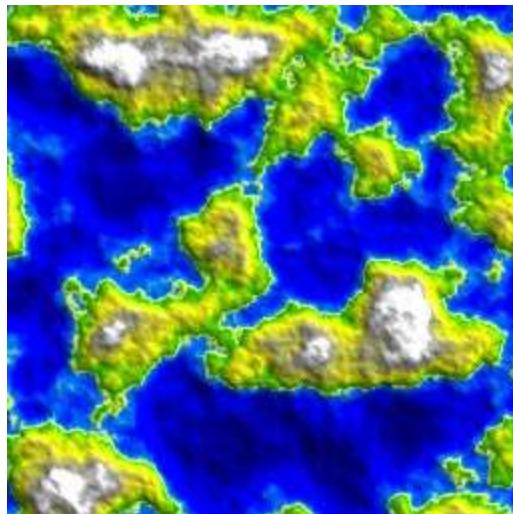


Figure 5.4.3 a

Comparing this to my implementation of Perlin noise, there is much more detail in this one, with colours blending more instead of being solid like EvSim.

Creating a map such as this is not as difficult as I had first imagined. My current method for drawing the map takes the generated Perlin noise result and draws colours based off of the returned number. For example, if the returned number is less than 0.3 it is displayed as deep water, which is a dark blue colour. To add detail, the program would take the RGB values, and adjust it to add variety in terrain colour. The terrains could also be blended, with two colours being interpolated to mix between the two.

A major consideration to take into account for this improvement is that the map needs to maintain clear terrain differences so that the user can easily differentiate between terrains. Furthermore, the changes in colours may also interfere with the colourblind accessibility, or even the current terrain choice. A boundary between the biomes would still have to be set so these issues are avoided.

If the improved map was to look like figure 5.4.3a, the program would visually look much better.

5.4.3.2 Organism appearance

As mentioned throughout the evaluation, and within limitations, I was not able to sufficiently display the organisms in a way that evolution is obvious. This has therefore become a major objective of future developments, perhaps even the first improvement I will make. There are two approaches I could make, the first through shapes and the second through images. While the size of organisms is already a factor which could be easily implemented, other attributes such as their preference to terrain could be displayed more accurately through images. On the other hand, attributes such as predation and sex could be displayed through shapes. Applying a mix of these two, potentially through a base layer image which has adaptive appearances, means the organisms are more accurately represented and displayed within EvSim.

5.4.4 Increasing the scope of evolution

Due to the fact that EvSim is an evolution simulator, I have to ensure that evolution is actually present and has a range of variation. One way of increasing the current level of variation, is to add in more terrains and attributes. Currently, there are 7 terrains (deep water, water, beach, grass, forest, mountain, and snow). A feature I would definitely want to add to EvSim is an increased range of terrains. Original plans for EvSim include a desert for example. The desert and other terrains were omitted from the design and development of EvSim due to time concerns and map limitations, however with more time available and a more profound knowledge and experience of the resources I have used, it would now be much easier to implement new terrains. Managing and keeping track of terrains throughout development (and the many changes made in between) was quite a difficult task, due to the fact that if one terrain changed, the others are likely to also need a change.

Now that the majority of EvSim and its processing is finalised, features such as an extended terrain pool, and a user selection of what biomes to create is now much more possible. While during development a new terrain would require a lot of adjustment, a new terrain at the end would only require additions to the terrains affect. Principally, a terrain is a value of what colour it is displayed as, meaning to add in a new terrain is the same as expanding an array, and creating a new colour. Deeper, adding in a new terrain would require an extension to the terrains package, and the attributes of the terrain defined and balanced.

Not only is the map a focus of future improvements, but also evolution itself. While the current evolution cycle includes 11 individual elements of organism development, this could easily be expanded and added to. For example, features such as amount of organisms birthed, or even a movement attribute could be added. Expanding the pool of attributes with EvSim would bring it closer to fulfilling its role as a ‘simulator’, with the amount of organisms birthed mimicking animals in real life such as frogs, which give birth to hundreds of children at a time (despite many dying), and a movement attribute mirroring animals such as birds, with the ability to fly. The terrain specific benefit (3.5.3) could also be expanded in more detail to include specifics about the organisms exception in that terrain. This would also allow the organisms to have more accurate depictions and representations through images, as described in 5.4.3.2. An organism which has a terrain specific benefit of being able to fly on land means that the organism would have wings displayed for example.

It is evident that increasing EvSim’s scope of evolution would benefit the simulation in many ways, with the most important being the fact that it does fulfil its role of being a simulator.

CODE REFERENCES

This section contains the full EvSim package, each class and file displayed here. For any references made to files within the code, refer to this section for that class in its final version. For files which change (such as backend JSON), there are examples of what would be stored, but keep in mind it is not constant.

This section is split, containing Processing, Models, Packages, and Storage. The Processing section contains the classes which are used to run EvSim, from the menus to the organism updates and the linking UI. The Models section contains encapsulated classes as well as the UI format itself (uncommented due to it being created by IntelliJ forms creator). The Packages section contains the classes which are part of an inheritance chain, used in processing, such as the food types and terrains. The Storage section contains JSON file examples, which are examples of what is held in the backend of EvSim. Keep in mind these are variable and will change during the operation of EvSim.

[This GitHub repository](#) also contains the source code for EvSim. The executable JAR file can be found in `EvSim/out/artifacts/EvSim_jar/EvSim.jar`. Installation of this JAR file alongside [Java Development Kit \(JDK\) 17](#) will allow EvSim to be executed on any machine compatible with Java.

C1 Processing

C1.1 Main.java

```
1. // This starts EvSim by loading the main menu
2. public class Main {
3.     public static void main(String[] args) {
4.         new OpeningMenu();
5.     } // Creates a new opening menu
6. }
```

C1.2 OpeningMenu.java

```
1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.io.File;
5. import java.io.IOException;
6. import java.io.InputStream;
7.
8. // This is the opening menu which is displayed to the user once they start the game
9. public class OpeningMenu extends JPanel {
10.     public static JFrame frame; // The window frame itself
11.     public static Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); // Gets
the users screen size
12.     public static int windowHeight = (int) screenSize.getHeight(); // Default value
13.     public static int windowWidth = (int) screenSize.getWidth(); // Default value
14.
15.     // The constructor for OpeningMenu which creates the side menu and buttons
```

```

16.     public OpeningMenu() {
17.         ObjectMapper objectMapper = new ObjectMapper(); // Initialising the JSON Object Mapper
18.         Options options;
19.
20.         // Attempts to read options.json
21.         try {
22.             File sourceFile = new File("options.json");
23.             if (sourceFile.exists()) { // If the file is within the source folder then read
from there
24.                 options = objectMapper.readValue(sourceFile, Options.class);
25.             } else {
26.                 // If the file doesn't exist in the source folder, read it from the JAR
resources
27.                 InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("options.json");
28.                 if (inputStream != null) { // If the file has been read successfully
29.                     options = objectMapper.readValue(inputStream, Options.class); // Pass the
JSON
30.                     inputStream.close();
31.                 } else {
32.                     throw new RuntimeException(); // Otherwise print an error
33.                 }
34.             }
35.         } catch (IOException e) {
36.             throw new RuntimeException(e); // Print error
37.         }
38.
39.         windowHeight = options.getWindowWidth(); // Gets the value stored at the key, and
returns as Integer
40.         windowHeight = options.getWindowHeight();
41.
42.         this.setSize(windowWidth, windowHeight); // Sets the JPanel size to what has changed
43.         this.setBackground(Color.black); // Sets the JPanel background colour to be black
44.
45.         frame = new JFrame("EvSim"); // Sets the title to be EvSim
46.         frame.setSize(windowWidth, windowHeight); // Sets the window size
47.         frame.setLocationRelativeTo(null); // Window location relative to nothing (top left /
full window)
48.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit the application if the
window is closed
49.         frame.setVisible(true); // Set it to be visible
50.
51.         JPanel menuPanel = createVerticalMenu(); // Creates a vertical side menu which
contains the buttons
52.         frame.add(this, BorderLayout.CENTER); // Adds the JPanel to the centre of the frame
53.         frame.add(menuPanel, BorderLayout.WEST); // Add the menu panel to the left side
54.
55.         frame.setVisible(true);
56.     }
57.
58.     // Creates the vertical menu containing the buttons for start, load, options and exit
59.     private JPanel createVerticalMenu() {
60.         JPanel menu = new JPanel(); // New JPanel to hold the buttons
61.         menu.setBackground(Color.darkGray); // Sets the background to be dark grey
62.         menu.setPreferredSize(new Dimension(100, getHeight())); // Sets the width to 100px and
the height to the same as the page
63.
64.         JButton startButton = new JButton("Start"); // Creates the JButtons
65.         JButton loadButton = new JButton("Load");
66.         JButton optionButton = new JButton("Options");
67.         JButton exitButton = new JButton("Exit");
68.
69.         setButtonProperties(startButton, "Starts a new iteration"); // Sets the styling to the
buttons
70.         setButtonProperties(loadButton, "Loads into a previous save");
71.         setButtonProperties(optionButton, "Set options for EvSim");
72.         setButtonProperties(exitButton, "Exit the program");
73.
74.         // Creates and adds an action listener from the ActionListener class
75.
76.         // Exits the currently running Java process

```

```

77.         exitButton.addActionListener(actionEvent -> System.exit(0));
78.
79.         // Starts a new simulation through the start game menu
80.         startButton.addActionListener(actionEvent -> {
81.             new StartGameMenu(); // Opens a new start menu
82.             frame.dispose(); // Closes the current frame
83.         });
84.
85.         // Opens the options menu
86.         optionButton.addActionListener(e -> {
87.             new MainMenuOptions(); // Creates a new options menu and displays
88.             frame.dispose(); // Removes the current frame
89.         });
90.
91.         loadButton.addActionListener(e -> new LoadSimulation()); // Creates and opens the load
menu
92.
93.         menu.add(startButton); // Add all buttons to the JPanel
94.         menu.add(loadButton);
95.         menu.add(optionButton);
96.         menu.add(exitButton);
97.
98.         return menu; // Return the JPanel to be added to the frame
99.     }
100.
101.    // This styles the buttons to be the same appearance
102.    private static void setButtonProperties(JButton button, String tooltip) { // Generic menu
button style
103.        button.setForeground(Color.black); // Sets the text colour to be black
104.        button.setBackground(Color.darkGray); // Sets the background button colour to be dark
gray (same as the menu)
105.        button.setToolTipText(tooltip); // Sets the help text that displays on mouse hover
106.        button.setPreferredSize(new Dimension(100, 50)); // Sets the button size
107.        button.setBorder(BorderFactory.createLineBorder(Color.darkGray, 1)); // Button border
to be same as background
108.        button.setFont(new Font("Arial", Font.BOLD, 16)); // The font and size of the text
109.    }
110.
111.    // Paints the EvSim logo at the centre (no matter the size of the window)
112.    @Override
113.    protected void paintComponent(Graphics g) {
114.        super.paintComponent(g);
115.        Dimension gameWindowSize = getSize(); // Gets the current window size as a Dimension
116.        FontMetrics fontMetrics = g.getFontMetrics(); // Class to get the characteristics of a
string of text
117.
118.        g.setColor(Color.white); // Sets the colour to white
119.        g.setFont(new Font("Agency FB", Font.BOLD, 108));
120.
121.        int textWidth = fontMetrics.stringWidth("EVSIM"); // Gets the width of the title
"EVSIM"
122.        int textHeight = fontMetrics.getHeight(); // Returns the height of the font (which
would be 108 in this case).
123.
124.        // variable for the x and y position of where the title should be.
125.        int titleTextX = (gameWindowSize.width - textWidth) / 2 - 100; // -100 to account for
the size of the menu on the left
126.        int titleTextY = (gameWindowSize.height - textHeight) / 2;
127.        // Takes the size of the game window, e.g. 1920, takes away the size of the text, e.g.
35,
128.        // and divides by 2 to get the exact centre on any window size.
129.
130.        g.drawString("EVSIM", titleTextX, titleTextY); // Draws the string to the page at
position x and y
131.
132.        g.setFont(new Font("Agency FB", Font.BOLD, 36));
133.        g.drawString("Press 'Start' to begin", titleTextX - 15 /*Left of title*/, titleTextY +
50 /*Below the title*/);
134.        // Draws the subtitle on the page just below and to the left of the title.
135.    }
136.}

```

C1.3 StartGameMenu.java

```
1. import com.fasterxml.jackson.databind.JsonNode;
2. import com.fasterxml.jackson.databind.ObjectMapper;
3. import javax.swing.*;
4. import java.awt.*;
5. import java.awt.event.MouseEvent;
6. import java.awt.event.MouseListener;
7. import java.io.File;
8. import java.io.IOException;
9. import java.io.InputStream;
10. import java.util.Random;
11.
12. // This class is the layout for the start game menu which allows the user to select their
options to set
13. // before starting the simulation
14. // The user can either start a simulation, return to the main menu, or preview the map from
this menu
15. public class StartGameMenu implements MouseListener {
16.     private JPanel panel1; // The panel containing the components below
17.     private JTextField seedTextField;
18.     private JTextField organismTextField;
19.     private JCheckBox allowSeasonsCheckBox; // Not used yet
20.     private JCheckBox allowPredatorPreyOrganismsCheckBox; // Not used yet
21.     private JButton startGameButton;
22.     private JButton backToHomeButton;
23.     private JTextField a1920TextField;
24.     private JTextField a1080TextField;
25.     private JButton previewButton;
26.     private final JFrame frame; // The window itself
27.     private final Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); // The
users monitor size
28.     public StartGameMenu() {
29.         ObjectMapper objectMapper = new ObjectMapper(); // Initialising the JSON Object Mapper
30.
31.         try {
32.             Options newData; // The options object which will be saved
33.             File sourceFile = new File("options.json"); // The file to save to
34.
35.             if (sourceFile.exists()) { // If it is present in the source folder then set the
object
36.                 newData = objectMapper.readValue(sourceFile, Options.class);
37.             } else {
38.                 // If the file doesn't exist in the source folder, read it from the JAR
resources
39.                 InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("options.json");
40.                 if (inputStream != null) {
41.                     newData = objectMapper.readValue(inputStream, Options.class);
42.                     inputStream.close();
43.                 } else {
44.                     throw new RuntimeException(); // Catch and print any errors
45.                 }
46.             }
47.
48.             // Gets the default window size from the JSON and sets it as the placeholder text
for the window size
49.             // of the start menu
50.             if(newData.getWindowWidth() >= (int) screenSize.getWidth())
a1920TextField.setText(String.valueOf((int) screenSize.getWidth()));
51.             else if(newData.getWindowWidth() <= (int) screenSize.getWidth())
a1920TextField.setText(String.valueOf(newData.getWindowWidth()));
52.             if(newData.getWindowHeight() >= (int) screenSize.getHeight())
a1080TextField.setText(String.valueOf((int) screenSize.getHeight()));
53.             else if(newData.getWindowHeight() <= (int) screenSize.getHeight())
a1080TextField.setText(String.valueOf(newData.getWindowHeight()));
54.         } catch (IOException e) {
55.             throw new RuntimeException(e); // Log any errors
56.         }
57.
58.         // Generate a new random seed
```

```

59.         seedTextField.setText(Double.toString(new Random().nextGaussian() * 255));
60.         PerlinNoise.setSeed(Double.parseDouble(seedTextField.getText())); // Set the seed in
Perlin noise
61.
62.         panel1.setPreferredSize(new Dimension(500,500)); // Set the preferred size of the
panel
63.
64.         startGameButton.addMouseListener(this); // Add required mouse listeners to buttons
65.         backToHomeButton.addMouseListener(this);
66.         previewButton.addMouseListener(this);
67.
68.         frame = new JFrame("Start Game"); // Set title of the window
69.         frame.setPreferredSize(new Dimension(500,500)); // Set size of the window
70.         frame.add(panel1); // As the panel already contains the other components, only the
panel has to be added to the frame.
71.         frame.setResizable(false);
72.         frame.pack();
73.         frame.setLocationRelativeTo(null);
74.         frame.setVisible(true);
75.     }
76.
77.     // This checks for any mouse clicks by the user
78.     // On components which have a mouse listener, such as the start game and return to home
buttons
79.     @Override
80.     public void mouseClicked(MouseEvent e) {
81.         // When they press on 'Start Game', the information should be validated.
82.         if(e.getSource() == startGameButton){
83.             String textInput = seedTextField.getText().trim(); // Gets the data from
seedTextField, and removes any blank space(.trim())
84.             String text2Input = organismTextField.getText().trim(); // Takes the number of
organisms and removes whitespace (.trim())
85.             String windowHeight = a1920TextField.getText().trim();
86.             String windowHeight = a1080TextField.getText().trim();
87.
88.             boolean invalidInput = false; // Tracks any invalidation so the game doesn't start
with invalid inputs
89.
90.             try {
91.                 // Map seed validation
92.                 double seedInput = Double.parseDouble(textInput); // Sends the trimmed data
from seedTextField into a double
93.                 if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
94.                     PerlinNoise.setSeed(seedInput); // Sets the seed in perlin noise
95.                 } else {
96.                     // If it is not in the bounds, then it displays the ErrorWindow, with html
styled text
97.                     new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a
number that is within the range of -1000.0 and 1000.0</p></html>");
98.                     invalidInput = true;
99.                 }
100.
101.             // Organism count validation
102.             int organismInput = Integer.parseInt(text2Input); // Passes it into an integer
103.             if (organismInput <= 300 && organismInput >= 1) { // Bounds
104.                 GenerateOrganisms.organismCount = organismInput; // Sets the organism
count to be generated
105.             } else {
106.                 // Display an error message
107.                 new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a
number that is within the range of 2 and 50</p></html>");
108.                 invalidInput = true;
109.             }
110.
111.             // Window size validation
112.             int windowHeightInput = Integer.parseInt(windowHeight); // Passing into
Integer
113.             int windowHeightInput = Integer.parseInt(windowWidth);
114.             if(windowHeightInput <= screenSize.getHeight() && windowHeightInput >= 500) {
// Bounds

```

```

115.                 windowHeight = Integer.toString(windowHeightInput); // Sets the window to
a validated Integer
116.             } else {
117.                 // Display an error message
118.                 new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a
window height between 500 and " + screenSize.getHeight() + "</p></html>");
119.                 invalidInput = true;
120.             }
121.             if(windowWidthInput <= screenSize.getWidth() && windowWidthInput >= 500) { // Bounds
122.                 windowHeight = Integer.toString(windowWidthInput); // Sets the window to a
validated Integer
123.             } else {
124.                 // display an error message
125.                 new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a
window width between 500 and " + screenSize.getWidth() + "</p></html>");
126.                 invalidInput = true;
127.             }
128.
129.         } catch (NumberFormatException error) { // Catch any passing errors
130.             // If there are any other errors, it gives the user the error message in html
format
131.             new ErrorWindow("<html><p style=\"text-align:center;\">Please provide valid
inputs.<br>One of your inputs contains an unexpected character</p></html>");
132.             invalidInput = true;
133.         }
134.
135.         // If there have been no invalid inputs, then load the game with the settings
136.         if(!invalidInput){
137.             GameWindow gameWindow = new GameWindow(Integer.parseInt(windowWidth),
Integer.parseInt(windowHeight), false, null);
138.             gameWindow.startGameThread(); // Creates a new game window from the game
window class, width and height as the input from the user
139.             frame.dispose(); // Removes the start game menu
140.             // Updates the users chosen options
141.             Organisms.predatorPreyEnabled =
allowPredatorPreyOrganismsCheckBox.isSelected();
142.             Organisms.seasonsEnabled = allowSeasonsCheckBox.isSelected();
143.         }
144.     }
145.
146.     // If the user desires to go back to home, display the start menu
147.     if(e.getSource() == backToHomeButton){
148.         new OpeningMenu(); // New opening menu
149.         frame.dispose(); // Remove the start menu
150.     }
151.
152.     // If they want to preview the map
153.     if(e.getSource() == previewButton){
154.         double seedInput = Double.parseDouble(seedTextField.getText()); // Parses the data
from seedTextField into a double
155.         // Validates the input and gets the desired window size
156.         if (seedInput >= -1000 && seedInput <= 1000) { // Bounds
157.             int width = Integer.parseInt(a1920TextField.getText()), height =
Integer.parseInt(a1080TextField.getText());
158.             PreviewMap(width, height); // Calls the preview map method to display what the
map will look like
159.         } else {
160.             // If it is not in the bounds, then it displays the ErrorWindow, with html
styled text
161.             new ErrorWindow("<html><p style=\"text-align:center;\">Please provide a seed
that is within the range of -1000.0 and 1000.0</p></html>");
162.         }
163.     }
164. }
165.
166. // The PreviewMap function allows the user to see the map before the simulation starts
167. // Paints only the map in the window
168. public void PreviewMap(int width, int height) {
169.     JFrame frame = new JFrame("Map Preview"); // New window
170.     // Offset of -250 to counter for what would usually be the side menu

```

```

171.         frame.setPreferredSize(new Dimension(width - 250, height)); // Set the size to be
equal to the map size
172.         frame.setLocation((int) (OpeningMenu.screenSize.getWidth() / 2) - width / 2, 0); //Middle of the screen
173.         frame.setResizable(false);
174.
175.         PerlinNoise.setSeed(Double.parseDouble(seedTextField.getText())); // Send to perlin
noise
176.         GenerateMap mapPanel = new GenerateMap(width + 250, height, false, null);
177.         frame.add(mapPanel); // Generate the map and add it to the window
178.
179.         frame.pack();
180.         frame.setVisible(true);
181.     }
182.
183.     // Not used, but required by the program
184.     @Override
185.     public void mousePressed(MouseEvent e) {} // Not needed
186.
187.     @Override
188.     public void mouseReleased(MouseEvent e) {} // Not needed
189.
190.     @Override
191.     public void mouseEntered(MouseEvent e) {} /// Not needed
192.
193.     @Override
194.     public void mouseExited(MouseEvent e) {} // Not needed
195. }
196.

```

C1.4 LoadSimulation.java

```

1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.MouseEvent;
5. import java.awt.event.MouseListener;
6. import java.io.File;
7. import java.io.IOException;
8. import java.nio.file.*;
9. import java.time.ZoneId;
10. import java.time.Instant;
11. import java.time.LocalDateTime;
12. import java.time.format.DateTimeFormatter;
13. import java.util.regex.Matcher;

```

```

14. import java.util.regex.Pattern;
15.
16. // This allows an old save to be loaded with the appropriate settings being carried over
17. // Code for linking the load menu as well as the load function itself (see mouseClicked)
18. public class LoadSimulation implements MouseListener {
19.     private JButton loadSaveButton;
20.     private JButton cancelButton;
21.     private JPanel panel1; // The panel containing the form itself
22.     private JList<String> list1;
23.     private JButton deleteButton;
24.     private JButton renameButton;
25.     public static JFrame frame; // The window for the load menu
26.
27.     // The constructor of LoadSimulation which displays the window itself
28.     // which contains the list of save files
29.     public LoadSimulation() {
30.         // This method returns a list model which holds all the names of the saves, as a
component which can be displayed on the JList
31.         String path = "./Save Games"; // Path to the parent folder
32.
33.         DefaultListModel<String> listModel = new DefaultListModel<>(); // List model storing
the save names and details
34.
35.         File folder = new File(path); // Parent folder
36.         File[] folders = folder.listFiles(File::isDirectory); // Array of all child folders
37.
38.         // Loops through folders and adds to list
39.         if (folders != null) {
40.             for (File subFolder : folders) {
41.                 // Create a date and time for the last modified date
42.                 LocalDateTime lastModifiedDateTime = LocalDateTime.ofInstant(
43.                     Instant.ofEpochMilli(subFolder.lastModified()), // Time of last
modification
44.                     ZoneId.systemDefault() // Time zone
45.                 );
46.                 // Bullet points and formats date last modified
47.                 listModel.addElement("<html><ul><li>" + subFolder.getName() + " Last Used: " +
lastModifiedDateTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy")) + "</li></ul></html>");
48.             }
49.         }
50.
51.         list1.setModel(listModel); // Add the filled model to the JList
52.
53.         renameButton.setEnabled(false); // Setting to be disabled by default, and adding mouse
listeners
54.         deleteButton.setEnabled(false);
55.         loadSaveButton.setEnabled(false);
56.         renameButton.addMouseListener(this); // MouseListeners to check if the user clicks on
the button
57.         deleteButton.addMouseListener(this);
58.         loadSaveButton.addMouseListener(this);
59.         cancelButton.addMouseListener(this);
60.         list1.addMouseListener(this);
61.
62.         frame = new JFrame("Load Simulation"); // The frame itself
63.         frame.setPreferredSize(new Dimension(500, 540)); // Ideal size
64.         frame.add(panel1); // panel1 contains all other components
65.         frame.setResizable(false); // So no size issues
66.         frame.pack(); // So all is correct size
67.         frame.setLocationRelativeTo(null); // Central
68.         frame.setVisible(true);
69.     }
70.
71.     // Checks for the user clicking on components which have a mouseListener
72.     // In this case the JList and the JButtons
73.     @Override
74.     public void mouseClicked(MouseEvent e) {
75.
76.         if(e.getSource() == list1){ // If the user clicks on an object in the list, enable the
buttons
77.             renameButton.setEnabled(true); // Allow the other buttons to become functional

```

```

78.         deleteButton.setEnabled(true);
79.         loadSaveButton.setEnabled(true);
80.     }
81.
82.     // Load the simulation under the selected file name
83.     if(e.getSource() == loadSaveButton && loadSaveButton.isEnabled()) {
84.         String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
85.         ObjectMapper objectMapper = new ObjectMapper();
86.         try {
87.             // Replace organisms.json with the stored json file
88.             Files.copy(Paths.get("./Save Games/" + currentlySelectedFile +
"/organisms.json"), Paths.get("./organisms.json"), StandardCopyOption.REPLACE_EXISTING);
89.             // Load the stored settings
90.             Settings newLoad = objectMapper.readValue(new File("./Save Games/" +
currentlySelectedFile + "/" + currentlySelectedFile + ".json"), Settings.class); // Creating an
array of organisms from organisms.json
91.             // start a new game. Sends loaded save as true to indicate what to load. Sends
the newLoad to be loaded
92.             if(GameWindow.frame != null) {
93.                 GameWindow.frame.dispose();
94.                 GameWindow.thread.interrupt();
95.             }
96.             GameWindow newGameWindow = new GameWindow(newLoad.getWindowWidth(),
newLoad.getWindowHeight(), true, newLoad);
97.             newGameWindow.startGameThread(); // Start the thread
98.             frame.dispose(); // Dispose of the load menu
99.             OpeningMenu.frame.dispose(); // Dispose of the opening menu
100.            } catch (IOException ex) {
101.                throw new RuntimeException(ex);
102.            }
103.        }
104.
105.        // Deletes a save (folder and JSON files)
106.        if(e.getSource() == deleteButton && deleteButton.isEnabled()) {
107.            String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
108.
109.            // Removes selected save from the list on the screen
110.            DefaultListModel<String> listModel = (DefaultListModel<String>) list1.getModel();
111.            listModel.remove(list1.getSelectedIndex());
112.
113.            // Deletes the folder itself
114.            try {
115.                // Deletes the folder and its contents recursively
116.                deleteFolder(Paths.get("./Save Games/" + currentlySelectedFile));
117.                frame.dispose(); // Refresh the load save page
118.                new LoadSimulation();
119.            } catch (IOException error) {
120.                throw new RuntimeException(error);
121.            }
122.        }
123.
124.        // Renames a save file
125.        if(e.getSource() == renameButton && renameButton.isEnabled()) {
126.            String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
127.
128.            // Gets the name that the user wishes to rename to, and renames in the class
renameFile
129.            new RenameFile(currentlySelectedFile);
130.        }
131.
132.        // Close the frame
133.        if(e.getSource() == cancelButton) {
134.            frame.dispose();
135.            GameWindow.gamePaused = false;
136.        }
137.    }
138.

```

```

139.     // The folder cannot be deleted if it has contents in it, therefore this method is needed
140.     to
141.     private static void deleteFolder(Path folderPath) throws IOException {
142.         // 'Walk' the file tree to delete each file starting from the given folder path
143.         Files.walk(folderPath).sorted((p1, p2) -> p1.compareTo(p2)).forEach(path -> {
144.             try {
145.                 Files.delete(path); // Deletes the sub files/folders, in this case
146.                     only the sub-files
147.             } catch (IOException e) {
148.                 throw new RuntimeException(e); // Log any errors
149.             }
150.         });
151.
152.     // Finds the name of the currently selected file from the JList
153.     private String currentSelectedFile(){
154.         String currentlySelectedFile = "";
155.         String htmlString = list1.getSelectedValue(); // The selected value (which returns an
156.             HTML string)
157.             // Uses a pattern and a matcher to extract the name of the file from the HTML string
158.             Pattern pattern = Pattern.compile("<li>(.*)? Last Used:")); // Finds the file name
159.                 amongst the last name and HTML data
160.                 Matcher matcher = pattern.matcher(htmlString);
161.                 if (matcher.find()) currentlySelectedFile = matcher.group(1); // If the pattern is
162.                     found then the file name can be updated
163.                     return currentlySelectedFile; // Return the name of the file
164.                 }
165.
166.             // Not used but still needed within the program
167.             @Override
168.             public void mousePressed(MouseEvent e) {}
169.
170.             @Override
171.             public void mouseReleased(MouseEvent e) {}
172.
173.             @Override
174.             public void mouseEntered(MouseEvent e) {}
175.
176.             @Override
177.             public void mouseExited(MouseEvent e) {}
178.     }

```

C1.5 MainMenuOptions.java

```

1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.ItemEvent;
5. import java.awt.event.ItemListener;
6. import java.awt.event.MouseEvent;
7. import java.awt.event.MouseListener;
8. import java.io.File;
9. import java.io.FileWriter;
10. import java.io.IOException;
11. import java.io.InputStream;
12. import java.util.ArrayList;
13. import java.util.List;
14.
15. // The options menu which displays the accessibility options such as colours and window size
16. public class MainMenuOptions implements ItemListener, MouseListener {
17.     private JPanel mainMenu; // The panel containing all other components which are also
17.     listed below
18.     private JComboBox grassColour;
19.     private JComboBox waterColour;
20.     private JComboBox mountainColour;
21.     private JComboBox beachColour;
22.     private JComboBox snowColour;
23.     private JComboBox forestColour;

```

```

24.     private JComboBox deepWaterColour;
25.     private JComboBox windowSize;
26.     private JButton backButton;
27.     private JButton saveButton;
28.     private JPanel grassPreview;
29.     private JPanel waterPreview;
30.     private JPanel mountainPreview;
31.     private JPanel beachPreview;
32.     private JPanel snowPreview;
33.     private JPanel forestPreview;
34.     private JPanel deepWaterPreview;
35.     JFrame optionsFrame; // The window which will contain the panel
36.     private final Dimension size = Toolkit.getDefaultToolkit().getScreenSize(); // Getting the
users monitor size
37.     private int newWindowHeight, newWindowWidth; // The new window size to set
38.     private final int maxWindowHeight = (int) size.getHeight(); // The maximum height and
width from the monitor size
39.     private final int maxWindowWidth = (int) size.getWidth();
40.     private boolean windowSizeChange = false; // Whether it has been changed or not
41.     // The terrain colours (similar to what is found in GenerateMap)
42.     private static final Color[] terrainColours = {Color.GREEN, Color.BLUE, Color.GRAY,
Color.YELLOW, Color.WHITE, new Color(13, 100, 10) /* DARK GREEN */, new Color(6, 11, 112) /* DARK
BLUE */};
43.
44.     // Constructor for the MainOptionsMenu
45.     // Adds the action listeners, sets the window, and updates the settings as needed
46.     public MainOptionsMenu() {
47.         grassColour.addItemListener(this); // Adds the relevant listeners
48.         waterColour.addItemListener(this); // Used to recognise user inputs and actions
49.         mountainColour.addItemListener(this);
50.         beachColour.addItemListener(this);
51.         forestColour.addItemListener(this);
52.         deepWaterColour.addItemListener(this);
53.         windowSize.addItemListener(this);
54.         backButton.addMouseListener(this);
55.         saveButton.addMouseListener(this);
56.
57.         updateCurrentSettings(); // Updates to what is stored in JSON
58.
59.         optionsFrame = new JFrame("Options");
60.         optionsFrame.setPreferredSize(new Dimension(357, 388)); // Same width as in the form
creator
61.         optionsFrame.add(mainMenu); // As the panel already contains the labels and drop-
downs, only the panel has to be added to the frame.
62.         optionsFrame.setResizable(false);
63.         optionsFrame.pack();
64.         optionsFrame.setLocationRelativeTo(null);
65.         optionsFrame.setVisible(true);
66.     }
67.
68.     // Ensures that what is stored in JSON is the same as what is displayed to the user
69.     private void updateCurrentSettings() {
70.         ObjectMapper objectMapper = new ObjectMapper();
71.         Options data;
72.         // Read the JSON data stored
73.         try {
74.             File sourceFile = new File("options.json"); // Adjust the path as needed
75.             if (sourceFile.exists()) {
76.                 data = objectMapper.readValue(sourceFile, Options.class);
77.             } else {
78.                 // If the file doesn't exist in the source folder, read it from the JAR
resources
79.                 InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("options.json");
80.                 if (inputStream != null) {
81.                     data = objectMapper.readValue(inputStream, Options.class);
82.                     inputStream.close();
83.                 } else {
84.                     throw new RuntimeException(); // Log any errors
85.                 }
86.             }

```

```

87.         List<Colours> colours = data.getColours(); // Returns the list of colours stored
in JSON
88.
89.         windowSize.setSelectedItem(data.getWindowWidth() + " x " +
data.getWindowHeight()); // The current set window size
90.             // Represented as "WindowWidth x WindowHeight", e.g. 1920 x 1080.
91.
92.             grassColour.setSelectedItem(colours.get(0).getSetting()); // Displaying the
current selected setting inn the options menu
93.             waterColour.setSelectedItem(colours.get(1).getSetting()); // Takes the object at
the first index, then takes the value of key "Setting" as a string
94.             mountainColour.setSelectedItem(colours.get(2).getSetting()); // get(2) returns the
object at array index 2
95.             beachColour.setSelectedItem(colours.get(3).getSetting()); //
get("Setting").asText() returns the current setting as a String
96.             forestColour.setSelectedItem(colours.get(4).getSetting()); // Which is then
displayed as the selected item on the menu
97.             deepWaterColour.setSelectedItem(colours.get(5).getSetting());
98.
99.     } catch (Exception exception) {
100.         throw new RuntimeException(exception); // Log any errors
101.     }
102.
103.     // If the users monitor is smaller than these sizes,
104.     // then remove them, so they cannot be selected
105.     // (would cause the window to cover the full screen - unusable)
106.     if (maxWindowHeight < 1080 || maxWindowWidth < 1920) windowSize.removeItem("1920 x
1080");
107.     if (maxWindowHeight < 2160 || maxWindowWidth < 3840) windowSize.removeItem("3840 x
2160");
108.     if (maxWindowHeight < 1440 || maxWindowWidth < 2560) windowSize.removeItem("2560 x
1440");
109.     if (maxWindowHeight < 768 || maxWindowWidth < 1366) windowSize.removeItem("1366 x
768");
110.     if (maxWindowHeight < 900 || maxWindowWidth < 1440) windowSize.removeItem("1440 x
900");
111.     if (maxWindowHeight < 720 || maxWindowWidth < 1280) windowSize.removeItem("1280 x
720");
112.     if (maxWindowHeight < 1024 || maxWindowWidth < 1280) windowSize.removeItem("1280 x
1024");
113. }
114.
115. // If the drop-down box changes state then update the menu accordingly
116. // Updates the JPanels showing the user the file currently selected.
117. @Override
118. public void itemStateChanged(ItemEvent e) {
119.     // These statements check which box was changed, and alters the relating JPanel
120.     // Overloaded and cannot be simplified in any way
121.     // Majority of this is just setting the colours which will be.
122.     // Refer to the comments on the side to know which colours it is
123.
124.     // If the user changed the grass drop down
125.     if (e.getSource() == grassColour) {
126.         if (grassColour.getSelectedItem() == "Default")
127.             grassPreview.setBackground(Color.GREEN); // Sets the colour for each colour-
blindness
128.
129.         else if (grassColour.getSelectedItem() == "Deuteranopia (1)")
130.             grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
131.         else if (grassColour.getSelectedItem() == "Deuteranopia (2)")
132.             grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
133.         else if (grassColour.getSelectedItem() == "Protanopia (1)")
134.             grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
135.         else if (grassColour.getSelectedItem() == "Protanopia (2)")
136.             grassPreview.setBackground(new Color(51, 51, 204)); // Purple-blue
137.         else if (grassColour.getSelectedItem() == "Tritanopia (1)")
138.             grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
139.         else if (grassColour.getSelectedItem() == "Tritanopia (2)")
140.             grassPreview.setBackground(new Color(255, 217, 0)); // Yellow
141.         else if (grassColour.getSelectedItem() == "Monochromacy (1)")
142.             grassPreview.setBackground(new Color(0, 0, 0)); // Black
142.         else if (grassColour.getSelectedItem() == "Monochromacy (2)")
```

```

143.         grassPreview.setBackground(new Color(255, 255, 255)); // White
144.     else if (grassColour.getSelectedItem() == "Trichromacy (1)")
145.         grassPreview.setBackground(new Color(255, 88, 51)); // Orange-red
146.     else if (grassColour.getSelectedItem() == "Trichromacy (2)")
147.         grassPreview.setBackground(new Color(0, 153, 0)); // Darker green
148.     terrainColours[0] = grassPreview.getBackground(); // Sets the grass (first index)
to be the colour of the panel.
149. }
150. // If the user changed the water drop down
151. else if (e.getSource() == waterColour) {
152.     if (waterColour.getSelectedItem() == "Default")
153.         waterPreview.setBackground(Color.BLUE); // Sets the colour for each colour-
blindness
154.     else if (waterColour.getSelectedItem() == "Deuteranopia (1)")
155.         waterPreview.setBackground(new Color(0, 128, 128)); // Teal
156.     else if (waterColour.getSelectedItem() == "Deuteranopia (2)")
157.         waterPreview.setBackground(new Color(255, 0, 0)); // Red
158.     else if (waterColour.getSelectedItem() == "Protanopia (1)")
159.         waterPreview.setBackground(new Color(0, 128, 128)); // Teal
160.     else if (waterColour.getSelectedItem() == "Protanopia (2)")
161.         waterPreview.setBackground(new Color(255, 0, 0)); // Red
162.     else if (waterColour.getSelectedItem() == "Tritanopia (1)")
163.         waterPreview.setBackground(new Color(0, 128, 128)); // Teal
164.     else if (waterColour.getSelectedItem() == "Tritanopia (2)")
165.         waterPreview.setBackground(new Color(255, 255, 0)); // Yellow
166.     else if (waterColour.getSelectedItem() == "Monochromacy (1)")
167.         waterPreview.setBackground(new Color(0, 0, 50)); // Dark Blue/Black
168.     else if (waterColour.getSelectedItem() == "Monochromacy (2)")
169.         waterPreview.setBackground(new Color(150, 150, 150)); // Light Grey
170.     else if (waterColour.getSelectedItem() == "Trichromacy (1)")
171.         waterPreview.setBackground(new Color(0, 128, 128)); // Teal
172.     else if (waterColour.getSelectedItem() == "Trichromacy (2)")
173.         waterPreview.setBackground(new Color(255, 0, 0)); // Red
174.     terrainColours[1] = waterPreview.getBackground(); // Sets the water (second index)
to be the colour of the panel
175. }
176. // If the user changed the mountain drop down
177. else if (e.getSource() == mountainColour) {
178.     if (mountainColour.getSelectedItem() == "Default")
179.         mountainPreview.setBackground(Color.GRAY); // Sets the colour for each colour-
blindness
180.     else if (mountainColour.getSelectedItem() == "Deuteranopia (1)")
181.         mountainPreview.setBackground(new Color(96, 96, 96)); // All of these are
shades of grey (to avoid biome blending), the higher the number in each, the lighter the grey
182.     else if (mountainColour.getSelectedItem() == "Deuteranopia (2)")
183.         mountainPreview.setBackground(new Color(160, 160, 160));
184.     else if (mountainColour.getSelectedItem() == "Protanopia (1)")
185.         mountainPreview.setBackground(new Color(50, 50, 50));
186.     else if (mountainColour.getSelectedItem() == "Protanopia (2)")
187.         mountainPreview.setBackground(new Color(150, 150, 150));
188.     else if (mountainColour.getSelectedItem() == "Tritanopia (1)")
189.         mountainPreview.setBackground(new Color(60, 60, 60));
190.     else if (mountainColour.getSelectedItem() == "Tritanopia (2)")
191.         mountainPreview.setBackground(new Color(140, 140, 140));
192.     else if (mountainColour.getSelectedItem() == "Monochromacy (1)")
193.         mountainPreview.setBackground(new Color(70, 70, 70));
194.     else if (mountainColour.getSelectedItem() == "Monochromacy (2)")
195.         mountainPreview.setBackground(new Color(130, 130, 130));
196.     else if (mountainColour.getSelectedItem() == "Trichromacy (1)")
197.         mountainPreview.setBackground(new Color(80, 80, 80));
198.     else if (mountainColour.getSelectedItem() == "Trichromacy (2)")
199.         mountainPreview.setBackground(new Color(120, 120, 120));
200.     terrainColours[2] = mountainPreview.getBackground(); // Sets the mountain (third
index) to be the colour of the panel
201. }
202. // If the user changed the beach drop down
203. else if (e.getSource() == beachColour) {
204.     if (beachColour.getSelectedItem() == "Default")
205.         beachPreview.setBackground(Color.YELLOW); // Sets the colour for each colour-
blindness
206.     else if (beachColour.getSelectedItem() == "Deuteranopia (1)"
```

```

207.         beachPreview.setBackground(new Color(128, 128, 0)); // yellow-brown
208.     else if (beachColour.getSelectedItem() == "Deuteranopia (2)")
209.         beachPreview.setBackground(new Color(0, 0, 255)); // light blue
210.     else if (beachColour.getSelectedItem() == "Protanopia (1)")
211.         beachPreview.setBackground(new Color(128, 100, 0)); // orange-brown
212.     else if (beachColour.getSelectedItem() == "Protanopia (2)")
213.         beachPreview.setBackground(new Color(0, 0, 200)); // dark blue
214.     else if (beachColour.getSelectedItem() == "Tritanopia (1)")
215.         beachPreview.setBackground(new Color(100, 100, 0)); // olive green
216.     else if (beachColour.getSelectedItem() == "Tritanopia (2)")
217.         beachPreview.setBackground(new Color(50, 50, 200)); // Blue-purple
218.     else if (beachColour.getSelectedItem() == "Monochromacy (1)")
219.         beachPreview.setBackground(new Color(90, 90, 90)); // dark grey
220.     else if (beachColour.getSelectedItem() == "Monochromacy (2)")
221.         beachPreview.setBackground(new Color(170, 170, 170)); // Light grey
222.     else if (beachColour.getSelectedItem() == "Trichromacy (1)")
223.         beachPreview.setBackground(new Color(100, 100, 50)); // Brown
224.     else if (beachColour.getSelectedItem() == "Trichromacy (2)")
225.         beachPreview.setBackground(new Color(100, 50, 255)); // Purple
226.     terrainColours[3] = beachPreview.getBackground(); // Sets the beach (fourth index)
to be the colour of the panel
227. }
228. // If the user changed the forest drop down
229. else if (e.getSource() == forestColour) {
230.     if (forestColour.getSelectedItem() == "Default")
231.         forestPreview.setBackground(new Color(13, 100, 10)); // Sets the colour for
each colour-blindness
232.     else if (forestColour.getSelectedItem() == "Deuteranopia (1)")
233.         forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
234.     else if (forestColour.getSelectedItem() == "Deuteranopia (2)")
235.         forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
236.     else if (forestColour.getSelectedItem() == "Protanopia (1)")
237.         forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
238.     else if (forestColour.getSelectedItem() == "Protanopia (2)")
239.         forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
240.     else if (forestColour.getSelectedItem() == "Tritanopia (1)")
241.         forestPreview.setBackground(new Color(106, 106, 106)); // Light grey
242.     else if (forestColour.getSelectedItem() == "Tritanopia (2)")
243.         forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
244.     else if (forestColour.getSelectedItem() == "Monochromacy (1)")
245.         forestPreview.setBackground(new Color(15, 15, 15)); // Lighter shade of black
246.     else if (forestColour.getSelectedItem() == "Monochromacy (2)")
247.         forestPreview.setBackground(new Color(205, 180, 255)); // Lilac
248.     else if (forestColour.getSelectedItem() == "Trichromacy (1)")
249.         forestPreview.setBackground(new Color(55, 55, 55)); // Dark Grey
250.     else if (forestColour.getSelectedItem() == "Trichromacy (2)")
251.         forestPreview.setBackground(new Color(150, 150, 150)); // Lighter Grey
252.     terrainColours[5] = forestPreview.getBackground(); // Sets the forest (fifth
index) to be the colour of the panel
253. }
254. // If the user changed the deep water drop down
255. else if (e.getSource() == deepWaterColour) {
256.     if (deepWaterColour.getSelectedItem() == "Default")
257.         deepWaterPreview.setBackground(new Color(6, 11, 112)); // Sets the colour for
each colour-blindness
258.     else if (deepWaterColour.getSelectedItem() == "Deuteranopia (1)")
259.         deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
260.     else if (deepWaterColour.getSelectedItem() == "Deuteranopia (2)")
261.         deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
262.     else if (deepWaterColour.getSelectedItem() == "Protanopia (1)")
263.         deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
264.     else if (deepWaterColour.getSelectedItem() == "Protanopia (2)")
265.         deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
266.     else if (deepWaterColour.getSelectedItem() == "Tritanopia (1)")
267.         deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
268.     else if (deepWaterColour.getSelectedItem() == "Tritanopia (2)")
269.         deepWaterPreview.setBackground(new Color(255, 255, 0)); // Yellow
270.     else if (deepWaterColour.getSelectedItem() == "Monochromacy (1)")
271.         deepWaterPreview.setBackground(new Color(30, 30, 30)); // Very dark grey
272.     else if (deepWaterColour.getSelectedItem() == "Monochromacy (2)")
273.         deepWaterPreview.setBackground(new Color(180, 255, 200)); // Mint Green

```

```

274.         else if (deepWaterColour.getSelectedItem() == "Trichromacy (1)")
275.             deepWaterPreview.setBackground(new Color(8, 81, 68)); // Teal
276.         else if (deepWaterColour.getSelectedItem() == "Trichromacy (2)")
277.             deepWaterPreview.setBackground(new Color(125, 14, 7)); // Red
278.         terrainColours[6] = deepWaterPreview.getBackground(); // Sets the deep water
(sixth index to be the colour of the panel
279.     }
280.     // SNOW is white so does not need to be changed
281.
282.     // Otherwise if the user changed the window size drop down
283.     if (e.getSource() == windowSize) {
284.         // Link the selected size
285.         if (windowSize.getSelectedItem() == "1920 x 1080") {
286.             if (newWindowHeight != 1080) windowSizeChange = true; // Show that the size
has been changed
287.             newWindowHeight = 1080; // Sets globally what the user wishes to change to
288.             newWindowWidth = 1920;
289.         } else if (windowSize.getSelectedItem() == "3840 x 2160") {
290.             if (newWindowHeight != 2160) windowSizeChange = true;
291.             newWindowHeight = 2160;
292.             newWindowWidth = 3840;
293.         } else if (windowSize.getSelectedItem() == "2560 x 1440") {
294.             if (newWindowHeight != 1440) windowSizeChange = true;
295.             newWindowHeight = 1440;
296.             newWindowWidth = 2560;
297.         } else if (windowSize.getSelectedItem() == "1366 x 768") {
298.             if (newWindowHeight != 768) windowSizeChange = true;
299.             newWindowHeight = 768;
300.             newWindowWidth = 1366;
301.         } else if (windowSize.getSelectedItem() == "1440 x 900") {
302.             if (newWindowHeight != 900) windowSizeChange = true;
303.             newWindowHeight = 900;
304.             newWindowWidth = 1440;
305.         } else if (windowSize.getSelectedItem() == "1280 x 720") {
306.             if (newWindowHeight != 720) windowSizeChange = true;
307.             newWindowHeight = 720;
308.             newWindowWidth = 1280;
309.         } else if (windowSize.getSelectedItem() == "1280 x 1024") {
310.             if (newWindowHeight != 1024) windowSizeChange = true;
311.             newWindowHeight = 1024;
312.             newWindowWidth = 1280;
313.         }
314.     }
315. }
316.
317. // This checks for any mouse clicks by the user
318. // On components which have a mouse listener, such as the save and cancel buttons
319. @Override
320. public void mouseClicked(MouseEvent e) {
321.     // Checks which button was pressed
322.     if (e.getSource() == saveButton) { // If it was the save button
323.         ObjectMapper objectMapper = new ObjectMapper(); // Initialising the object mapper
324.
325.         if (newWindowHeight == 0) { // If the window height was not changed, or is 0
326.             // Ensure the window height is not saved as 0
327.             newWindowHeight = 1080;
328.             windowSizeChange = true;
329.         }
330.         if (newWindowWidth == 0) { // If the window width was not changed, or is 0
331.             // Ensure the window width is not saved as 0
332.             newWindowWidth = 1920;
333.             windowSizeChange = true;
334.         }
335.
336.         // Checking that the window size can fit
337.         if (newWindowHeight <= maxWindowHeight && newWindowWidth <= maxWindowWidth &&
windowSizeChange) {
338.             // Attempt to save to the JSON
339.             try {
340.                 Options newData; // The options object which will be saved
341.                 File sourceFile = new File("options.json"); // The file to save to

```

```

342.
343.             if (sourceFile.exists()) { // If it is present in the source folder then
344.                 newData = objectMapper.readValue(sourceFile, Options.class);
345.             } else {
346.                 // If the file doesn't exist in the source folder, read it from the
347.                 JAR resources
348.                 InputStream inputStream =
349.                     getClass().getClassLoader().getResourceAsStream("options.json");
350.                     if (inputStream != null) {
351.                         newData = objectMapper.readValue(inputStream, Options.class);
352.                         inputStream.close();
353.                     } else {
354.                         throw new RuntimeException(); // Catch and print any errors
355.                     }
356.             }
357.             newData.setWindowHeight(newWindowHeight); // Changing the values, first
358.             part is the key to change, second is what to change the value to
359.             newData.setWindowWidth(newWindowWidth);

360.             // Updating what is selected by the user with what will be stored
361.             int colourCount = 0;
362.             List<Colours> newColours = new ArrayList<>(); // The list of new colours
363.             to set
364.             // For each colour in terrainColours
365.             // Terrain colours was updated to the same as what was displayed in the
366.             coloured JPanel
367.             for (Color color : terrainColours) {
368.                 Colours newColour = new Colours(); // The new colour
369.                 newColour.setRed(color.getRed()); // Adding the red, green and blue
370.                 values for each colours into individual objects
371.                 newColour.setGreen(color.getGreen());
372.                 newColour.setBlue(color.getBlue());

373.                 if (colourCount == 0) newColour.setSetting((String)
374. grassColour.getSelectedItem());
375.                     else if (colourCount == 1) newColour.setSetting((String)
376. waterColour.getSelectedItem());
377.                         else if (colourCount == 2) newColour.setSetting((String)
378. mountainColour.getSelectedItem());
379.                             else if (colourCount == 3) newColour.setSetting((String)
380. beachColour.getSelectedItem());
381.                                 else if (colourCount == 4) newColour.setSetting((String)
382. forestColour.getSelectedItem());
383.                                     else if (colourCount == 5) newColour.setSetting((String)
384. deepWaterColour.getSelectedItem());
385.                                         newColours.add(newColour); // Adding it to the object array
386.                                         colourCount++; // Which colour is being set
387.                                         }

388.             newData.setColours(newColours); // Writing to the new JSON

389.             String newJSON = objectMapper.writeValueAsString(newData); // Changing to
390.             a string, so it can be written to the file
391.             FileWriter fileWriter = new FileWriter("options.json");
392.             fileWriter.write(newJSON); // Writing to the file
393.             fileWriter.close(); // No longer needed
394.             } catch (IOException ex) {
395.                 throw new RuntimeException(ex); // Log any errors
396.             }

397.             new OpeningMenu(); // Return to the opening menu
398.             optionsFrame.dispose(); // Dispose of the current frame
399.             }

399.         } else {
400.             // If the window size is larger than the screen, then the appropriate error
401.             message is displayed

```

```

396.             new ErrorWindow("<html><p style=\"text-align:center;\">That size is not
available for your monitor.<br>Please select a smaller size<br>Monitor Size: " + maxWindowWidth + "
x " + maxWindowHeight + "</p></html>");
397.         }
398.     }
399.
400.     // If the user does not wish to save any changes
401.     if (e.getSource() == backButton) {
402.         new OpeningMenu(); // Return to the opening menu
403.         optionsFrame.dispose(); // Ignores any inputs, and closes the menu
404.     }
405. }
406.
407. // Not used but still needed to be present in the program
408. @Override
409. public void mousePressed(MouseEvent e) {} // Not Used
410.
411. @Override
412. public void mouseReleased(MouseEvent e) {} // Not Used
413.
414. @Override
415. public void mouseEntered(MouseEvent e) {} // Not Used
416.
417. @Override
418. public void mouseExited(MouseEvent e) {} // Not Used
419. }

```

C1.6 GameWindow.java

```

1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.MouseEvent;
5. import java.awt.event.MouseListener;
6. import java.io.File;
7. import java.io.IOException;
8.
9. // This is where the simulation itself runs within.

```

```

10. // Paints the map and organisms, calls the organisms to update, and updates the side menu
11. public class GameWindow extends JPanel implements Runnable, MouseListener {
12.     public static JFrame frame; // The window it is contained in
13.     public static Thread thread; // To have other processes running such as the updating of
organisms in a run() method
14.     public static int years = 0; // The current year of the program
15.     public static int timeFactor = 2500; // The time in ms to execute each year
16.     public static int WIDTH, HEIGHT; // The size of the window
17.     private final GenerateMap mapPanel; // The map
18.     public static boolean gamePaused = false; // Whether the simulation is paused
19.     private final SideMenu sm; // The side menu
20.     public static int organismSelected = -1; // The currently selected organism, -1 by default
meaning unselected
21.     public static Organism[] organisms; // The array of organisms currently being operated
22.     private final ObjectMapper objectMapper = new ObjectMapper(); // Jackson's object mapper
to read JSON
23.     private static final JTextArea consoleTextArea = new JTextArea(); // The console text
24.     private static final JScrollPane consoleScrollPane = new JScrollPane(consoleTextArea); // Surround the text with a scroll pane
25.     public static JPanel consolePanel = new JPanel(new BorderLayout()); // The console panel
itself
26.     public static boolean showConsole; // Whether to show the console panel or not
27.     // Constructor for GameWindow, takes in window size, whether it is to load a save or not
and the save file itself if applicable
28.     public GameWindow(int Width, int Height, boolean LoadedSave, Settings newLoad) {
29.         WIDTH = Width; // Updating the global window size
30.         HEIGHT = Height;
31.         frame = new JFrame("EvSim"); // The title of the window
32.         frame.setPreferredSize(new Dimension(WIDTH, HEIGHT)); // The size to set
33.         frame.setLocation((int) (OpeningMenu.screenSize.getWidth() / 2) - Width / 2, 0); // Places it in the centre top of the screen
34.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit the program if this
window is closed
35.         frame.setResizable(false); // So there are no size/map issues the window size is fixed
36.
37.         if(!LoadedSave) { // If it is not a loaded save, start a new one
38.             years = 0;
39.             mapPanel = new GenerateMap(Width, Height, false, null);
40.             frame.add(mapPanel); // Create and add the map to the window
41.
42.             GenerateOrganisms organismGeneration = new GenerateOrganisms(Width, Height);
43.             frame.add(organismGeneration); // Create and add the organisms to the window
44.         } else { // If it is a loaded save
45.             years = newLoad.getCurrentYear(); // Set the years to be the same as stored
46.             PerlinNoise.setSeed(newLoad.getSeed()); // Set the map seed to paint as the one
stored
47.             Organisms.season = newLoad.getCurrentSeason(); // Set all boolean variables to be
the same as stored
48.             Organisms.seasonsEnabled = newLoad.isSeasonsEnabled();
49.             Organisms.predatorPreyEnabled = newLoad.isPredatorPreyEnabled();
50.             mapPanel = new GenerateMap(newLoad.getWindowWidth(), newLoad.getWindowHeight(),
true, newLoad);
51.             frame.add(mapPanel); // Create and add the map (from the stored seed)
52.         }
53.
54.         update(); // Ensuring organisms contains what was written in JSON before it needs to
be used
55.
56.         consoleTextArea.setEditable(false); // So that the user cannot type in the console
57.         consoleLog("New Simulation Started!"); // Displaying a welcome message in the console
58.
59.         frame.add(this);
60.         sm = new SideMenu();
61.         frame.add(sm.getSideMenu(), BorderLayout.WEST); // Create and add the side menu on the
left
62.         frame.addMouseListener(this); // So any mouse clicks are recognised
63.         frame.pack();
64.         frame.setVisible(true);
65.     }
66.
67.     public void startGameThread(){ // Starts the execution

```

```

68.         thread = new Thread(this);
69.         thread.start();
70.     }
71.     @Override
72.     public void run() { // This controls the speed at which the simulation runs
73.         int FPS = 60;
74.         double drawInterval = (double) 1000000000 / FPS; // Time interval between desired
frames
75.         double delta = 0;
76.         long lastTime = System.nanoTime();
77.         long currentTime;
78.         long timer = 0;
79.
80.         while (thread != null) { // If the thread is started
81.             currentTime = System.nanoTime();
82.             delta += (currentTime - lastTime) / drawInterval; // Time between last frame and
delta
83.             timer += (currentTime - lastTime); // Time since last frame
84.             lastTime = currentTime;
85.             if(delta >= 1) { // When to update
86.                 try {
87.                     if (!gamePaused && organisms != null) { // If the game is paused, and the
organisms variable has data
88.                         new Organisms(); // Updates all the organisms
89.                         update(); // Calls the update method (to sync JSON and organisms
variable)
90.                         years++; // Update the years
91.                     }
92.                     Thread.sleep(timeFactor); // 1000 is every second, 10000 is every 10
seconds
93.                 } catch (InterruptedException e) {
94.                     System.out.println();
95.                 }
96.                 repaint(); // Repaints the paintComponent
97.                 delta--;
98.             }
99.             if(timer >= 1000000000) { // Checks to see if a second has passed
100.                 timer = 0; // resets the timer
101.             }
102.         }
103.     }
104.     public void update() { // Syncs the JSON with the organisms variable
105.         try {
106.             organisms = objectMapper.readValue(new File("organisms.json"), Organism[].class);
// Creating an array of organisms from organisms.json
107.             //objectMapper.readValue(a,b) takes the data from the file (a), and parses it into
the class array (b)
108.         } catch (IOException e) {
109.             throw new RuntimeException(e); // Log any errors
110.         }
111.     }
112.     public void paintComponent(Graphics g) { // Painting the organisms and the map
113.         super.paintComponent(g);
114.
115.         mapPanel.draw(g, WIDTH, HEIGHT); // Draw the map
116.         if(organisms != null) GenerateOrganisms.draw(g); // Draw the organisms
sm.UpdateUI(); // Update the side menu
117.
118.         if(!showConsole) remove(consolePanel); // If the user doesn't want to see the console,
remove it
119.         else {
120.             consolePanel.add(consoleScrollPane); // Otherwise add in the console
121.             consolePanel.setPreferredSize(new Dimension(300, 100)); // Sets the size
122.             add(consolePanel); // Adds to the frame
123.         }
124.     }
125. }
126.
127. // This adds a message to the console, can be called from anywhere
128. public static void consoleLog(String message) { // Takes a string input and adds to the
console

```

```

129.         consoleTextArea.append(message + "\n"); // Add to the text area, and allow for the
next line to be added to through \n
130.         SwingUtilities.invokeLater(() -> { // Sets the scroll bar to be at the bottom to
always show most recent messages
131.             JScrollBar verticalScrollBar = consoleScrollPane.getVerticalScrollBar();
132.             verticalScrollBar.setValue(verticalScrollBar.getMaximum());
133.         });
134.     }
135.
136.     // This checks whether the mouse has been clicked (up and down in short succession)
137.     @Override
138.     public void mouseClicked(MouseEvent mouseEvent) {
139.         // Gets the mouse location in relation to the map (offset by 260 and 35
140.         int newMouseX = mouseEvent.getX() - 260, newMouseY = mouseEvent.getY() - 35;
141.         Organism[] newOrganisms; // A separate variable read from the JSON so it is not
dependant on the update() being completed
142.         try {
143.             newOrganisms = objectMapper.readValue(new File("organisms.json"),
Organism[].class); // Creating an array of organisms from organisms.json
144.             for (Organism organism : newOrganisms) { // For each organism (object) in
Organisms (array)
145.                 int tempOrganismX = organism.getxCoordinate(), tempOrganismY =
organism.getyCoordinate(); // Setting the current organisms coordinates into variables
146.                 int xDifference = Math.abs(newMouseX - tempOrganismX); // Returning the
difference between mouse and organism coordinates
147.                 int yDifference = Math.abs(newMouseY - tempOrganismY); // Returns as an
absolute number, so it can be compared to only one bound
148.
149.                 // If both are within the bound of 10
150.                 if(xDifference <= 10 && yDifference <= 10) {
151.                     if(organism.getStats().isDead() && GenerateOrganisms.showDeadOrganisms){
152.                         // Updating both boolean variables which are to be checked
153.                         organismSelected = organism.getOrganismId();
154.                         sm.UpdateUI();
155.                         break; // To ensure organismSelected is not replaced
156.                     } else if (!organism.getStats().isDead() &&
!GenerateOrganisms.showDeadOrganisms){ // Ensures the user cannot select an organism which is dead
if dead organisms are not showing
157.                         // Updating both boolean variables which are to be checked
158.                         organismSelected = organism.getOrganismId();
159.                         sm.UpdateUI();
160.                         break; // To ensure organismSelected is not replaced
161.                     }
162.                 } else {
163.                     organismSelected = -1; // To ensure all other organisms are not selected
164.                     if(organism == newOrganisms[newOrganisms.length - 1]) sm.UpdateUI(); // If
the user has clicked away from an organism, update to show global statistics
165.                 }
166.             }
167.         } catch (IOException e) {
168.             throw new RuntimeException(e); // Log any errors
169.         }
170.     }
171.
172.     // Not used, but required to be present
173.     @Override
174.     public void mousePressed(MouseEvent mouseEvent) {}
175.
176.     @Override
177.     public void mouseReleased(MouseEvent mouseEvent) {}
178.
179.     @Override
180.     public void mouseEntered(MouseEvent mouseEvent) {}
181.
182.     @Override
183.     public void mouseExited(MouseEvent mouseEvent) {}
184. }
```

C1.7 GenerateMap.java

```
1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.io.File;
5. import java.io.InputStream;
6. import java.util.List;
7.
8. // This is how the map is generated
9. // Uses the PerlinNoise class to generate a random terrain
10. public class GenerateMap extends JPanel {
11.     public double scale = 100; // This can be changed / adjusted to change the size of the map
that is generated
12.     public static double[][] map; // The map array
13.     // The default colours to show
14.     public static Color[] terrainColors = {Color.GREEN, Color.BLUE, Color.GRAY, Color.YELLOW,
Color.WHITE, new Color(13, 100, 10) /* DARK GREEN */, new Color(6, 11, 112) /* DARK BLUE */};
15.
16.     // The constructor for GenerateMap which creates the map itself
17.     public GenerateMap(int Width, int Height, boolean LoadedSave, Settings newLoad) {
18.         setLayout(null); // So there is no layout which may cause an issue painting
19.         Width = Width - 250; // Offset the width (to count for the side menu)
20.         readJSONColours(); // Updates the terrainColours with what is stored in JSON
21.         map = new double[Width][Height]; // Sets the size to be equal to the size of the map
22.         PerlinNoise noiseGenerator;
23.         if (!LoadedSave) noiseGenerator = new PerlinNoise(); // Initialises the perlin noise
generator if it is a new simulation
24.         else noiseGenerator = new PerlinNoise(newLoad.getSeed()); // Else initialised with the
set seed
25.         for (int y = 0; y < Height; y++) { // Paint along the x and y axis of the map
26.             for (int x = 0; x < Width; x++) {
27.                 // Casts coordinates to generate noise
28.                 double nx = (double) x / Width;
29.                 double ny = (double) y / Height;
```

```

30.             double terrainValue = noiseGenerator.noise(nx * scale, ny * scale);
31.             double normalisedValue = (terrainValue + 1.0) / 2.0; // Makes the generated
value between 0 and 1
32.             map[x][y] = normalisedValue; // Adding it to the map array
33.         }
34.     }
35. }
36.
37. // Reads and updates the colour array with what is stored in JSON
38. private void readJSONColours() {
39.     ObjectMapper objectMapper = new ObjectMapper(); // Initializing the Jackson JSON
Object Mapper
40.     Options options;
41.     // Try to read from the source folder
42.     try {
43.         File sourceFile = new File("options.json");
44.         if (sourceFile.exists()) { // If the file is within the source folder then read
from there
45.             options = objectMapper.readValue(sourceFile, Options.class);
46.         } else {
47.             // If the file doesn't exist in the source folder, read it from the JAR
resources
48.             InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("options.json");
49.             if (inputStream != null) { // If the file has been read successfully
50.                 options = objectMapper.readValue(inputStream, Options.class); // Pass the
JSON
51.                 inputStream.close();
52.             } else {
53.                 throw new RuntimeException(); // Otherwise print an error
54.             }
55.         }
56.
57.         List<Colours> coloursList = options.getColours(); // The list of colours
58.
59.         int counter = 0;
60.         for (Colours colours : coloursList) { // Updating each colour to the
terrainColours array
61.             terrainColors[counter] = new Color(colours.getRed(), colours.getGreen(),
colours.getBlue());
62.             counter++;
63.         }
64.     } catch (Exception e) {
65.         throw new RuntimeException(e); // Any issues output the error
66.     }
67. }
68.
69. // This returns the terrain type as a string (to be set as the current terrain of the
organism for example
70. public static String getTerrain(int x, int y) {
71.     double terrainType = map[x][y]; // The terrain type at the current x and y coordinates
72.     Color currentTerrain = getColor(terrainType); // Returns the colour at that position
73.     if (currentTerrain == terrainColors[0]) return "Grass"; // Return the terrain name
74.     else if (currentTerrain == terrainColors[1]) return "Water";
75.     else if (currentTerrain == terrainColors[2]) return "Mountain";
76.     else if (currentTerrain == terrainColors[3]) return "Beach";
77.     else if (currentTerrain == terrainColors[4]) return "Snow";
78.     else if (currentTerrain == terrainColors[5]) return "Forest";
79.     else return currentTerrain == terrainColors[6] ? "Deep Water" : "Default";
80. }
81.
82. // Two draw methods as the way that the preview map and the simulation work together is
different.
83. public void draw(Graphics g, int Width, int Height) { // Used during the simulation as
there are multiple components
84.     Width = Width - 250; // Offset the width to count for the side menu
85.     for (int row = 0; row < Width; row++) {
86.         for (int col = 0; col < Height; col++) {
87.             double terrainType = map[row][col];
88.             Color terrainColor = getColor(terrainType); // Sets the correct colour based
on the value generated from noise at coordinates row, col

```

```

89.             g.setColor(terrainColor);
90.             g.fillRect(row, col, 1, 1); // Paints the map to the screen
91.         }
92.     }
93. }
94.
95. @Override
96. protected void paintComponent(Graphics g) { // Used during the map preview as only the map
needs to be shown
97.     int Width = getWidth(); // Counter the space usually taken by the side menu
98.     int Height = getHeight();
99.     super.paintComponent(g);
100.
101.    for (int row = 0; row < Width; row++) { // Painting the map
102.        for (int col = 0; col < Height; col++) {
103.            double terrainType = map[row][col];
104.            Color terrainColor = getColor(terrainType); // Sets the correct colour based
on the value generated from noise at coordinates row, col
105.            g.setColor(terrainColor);
106.            g.fillRect(row, col, 1, 1); // Paints them to the screen
107.        }
108.    }
109. }
110.
111. // The terrain type (colour) that is to be painted depending on the double value generated
by Perlin Noise
112. private static Color getColor(double terrainType) { // Defines how the biomes are
generated.
113.     Color terrainColor;
114.     if (terrainType < 0.3) {
115.         terrainColor = terrainColors[6]; // Perlin noise can only generate numbers between
0 and 1, anything that is generated below 0.3 will be deep water.
116.     } else if (terrainType < 0.5) {
117.         terrainColor = terrainColors[1]; // Anything below 0.5 will be water
118.     } else if (terrainType < 0.55) {
119.         terrainColor = terrainColors[3]; // Anything below 0.55 will be desert
120.     } else if (terrainType < 0.6) {
121.         terrainColor = terrainColors[0]; // Anything below 0.6 will be grass
122.     } else if (terrainType < 0.65) {
123.         terrainColor = terrainColors[5]; // Anything below 0.65 will be forest
124.     } else if (terrainType < 0.7) {
125.         terrainColor = terrainColors[0]; // Anything below 0.7 will be grass
126.     } else if (terrainType < 0.8) {
127.         terrainColor = terrainColors[2]; // Anything below 0.8 will be mountains
128.     } else terrainColor = terrainColors[4]; // Anything above that will be snow
129.     return terrainColor; // Returns the colour
130. }
131. }

```

C1.8 GenerateOrganisms.java

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.io.File;
4. import java.util.Random;
5. import com.fasterxml.jackson.databind.DeserializationFeature;
6. import org.json.simple.JSONArray;
7. import org.json.simple.JSONObject;
8. import java.io.IOException;
9. import java.io.FileWriter;
10. import com.fasterxml.jackson.databind.ObjectMapper;
11. public class GenerateOrganisms extends JPanel {
12.     public static int organismCount = 50; // Default is 50, can be changed
13.     private final Random random = new Random(); // For random attributes
14.     private int x; // The current coordinates
15.     private int y;
16.     public static int organismRadius = 1; // Default is 1, can be changed
17.     public static boolean showDeadOrganisms = false; // Whether to paint dead organisms or not
18.     @SuppressWarnings("unchecked")
19.     public GenerateOrganisms(int width, int height) {

```

```

20.         width = width - 300; // Counter the side menu and place them away from the edges
21.         height = height - 35;
22.         String filename = "organisms.json"; // The file for the program to write to
23.         JSONArray jsonArray = new JSONArray(); // Creates a new JSON Array (which will contain
objects/organisms)
24.
25.         int tempSex = 0; // The default value for the organisms sex
26.
27.         for (int organismCounter = 0; organismCounter < organismCount; organismCounter++) {
28.             JSONObject jsonObject = new JSONObject(); // Creates a new JSON object for each
organism
29.             newOrganismLocation(width, height); // generates new x and y coordinates
30.
31.             // Adds a key/value pair to the json object initialised above
32.             jsonObject.put("organismId", organismCounter);
33.             jsonObject.put("xCoordinate", x);
34.             jsonObject.put("yCoordinate", y);
35.             jsonObject.put("currentTerrain", GenerateMap.getTerrain(x, y));
36.
37.             JSONObject statsObject = getStatsObject();
38.
39.             jsonObject.put("stats", statsObject);
40.             if(organismCount == 2) {
41.                 jsonObject.put("sex", tempSex); // Ensures that if there are 2 organisms, one
is male and one is female
42.                 tempSex = 1;
43.             } else jsonObject.put("sex", random.nextInt(0, 2)); // Random between male and
female
44.
45.             jsonObject.put("fatherID", -1);
46.             jsonObject.put("motherID", -1);
47.             jsonObject.put("predator", false);
48.
49.             jsonArray.add(jsonObject); // Adding the individual objects to the array
50.         }
51.         // Write the JSON array to the file
52.         try (FileWriter fileWriter = new FileWriter(filename)) {
53.             fileWriter.write(jsonArray.toJSONString());
54.             fileWriter.flush();
55.             //System.out.println("JSON array has been written to " + filename);
56.         } catch (IOException e) {
57.             e.printStackTrace();
58.         }
59.     }
60.     @SuppressWarnings("unchecked")
61.     private JSONObject getStatsObject() {
62.         JSONObject statsObject = new JSONObject(); // The stats object is initialised
63.         statsObject.put("currentAwareness", 1.0); // Theses are all the stats to be stored as
well as their default values
64.         statsObject.put("organismsEncountered", 0.0); // Refer to documentation for
explanation of these values and their purposes
65.         statsObject.put("awarenessHistory", new JSONArray());
66.         statsObject.put("age", 0);
67.         statsObject.put("preferredFood", null);
68.         statsObject.put("preferredFoodString", null);
69.         statsObject.put("eatenFood", null);
70.         statsObject.put("recentFood", new JSONArray());
71.         statsObject.put("foodFactor", 0.0);
72.         statsObject.put("sizeFactor", 0.0);
73.         statsObject.put("size", 0.5);
74.         statsObject.put("speed", 5.0);
75.         statsObject.put("startSpeed", 5.0);
76.         statsObject.put("currentTemp", 0);
77.         statsObject.put("preferredTemp", 0);
78.         statsObject.put("recentTemp", new JSONArray());
79.         statsObject.put("tempFactor", 1);
80.         statsObject.put("preferredTerrain", "Default");
81.         statsObject.put("secondPreferredTerrain", "Default");
82.         statsObject.put("thirdPreferredTerrain", "Default");
83.         statsObject.put("health", 100.0);
84.         statsObject.put("dead", false);

```

```

85.         statsObject.put("ageLastBred", 15);
86.         statsObject.put("generation", 1);
87.         statsObject.put("organismsMet", 0);
88.         statsObject.put("totalOrganismsMet", 0);
89.         statsObject.put("tsb", null);
90.         statsObject.put("lastAttack", 10);
91.         return statsObject; // Returns the stats as a JSONArray
92.     }
93.
94.     // Generates a random new organism location in a circle at the centre of the map
95.     private void newOrganismLocation(int Width, int Height) {
96.         int epicenterX = Width / 2; // Calculate where the centre of the epicentre should be
(middle of the map)
97.         int epicenterY = Height / 2;
98.
99.         // Set the epicenter radius to be at most one-fourth of the map width or preferably 5
times the number of organisms
100.        int epicenterRadius = Math.min(Width / 4, organismCount * 5);
101.
102.        // Generate random coordinates for each organism
103.        for (int i = 0; i < organismCount; i++) {
104.            // Generate a random angle in radians
105.            double angle = 2 * Math.PI * random.nextDouble();
106.
107.            // Generate a random distance within the epicenter radius
108.            double distance = epicenterRadius * Math.sqrt(random.nextDouble());
109.
110.            // Calculate the x and y coordinates for the organism based on the polar
coordinates
111.            // Uses trigonometry to convert polar coordinates (angle and distance ^) to
cartesian coordinates (x and y - what this program uses)
112.            int organismX = (int) (epicenterX + distance * Math.cos(angle));
113.            int organismY = (int) (epicenterY + distance * Math.sin(angle));
114.
115.            // Ensure the organism is within the map boundaries
116.            x = Math.max(0, Math.min(Width - 1, organismX));
117.            y = Math.max(0, Math.min(Height - 1, organismY));
118.        }
119.    }
120.
121.    // Draws the organisms onto the map each year (is called by GameWindow)
122.    public static void draw(Graphics g) {
123.        Graphics2D g2d = (Graphics2D) g; // Creates a graphics 2d component
124.        ObjectMapper objectMapper = new ObjectMapper();
125.        objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_ARRAY_AS_NULL_OBJECT,
true); // To avoid any null errors
126.        objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT,
true);
127.        File file = new File("organisms.json"); // The file to read
128.        if (file.length() > 0) { // If the file is not complete / empty then don't paint
anything (as an error will be returned)
129.            for (Organism organism : GameWindow.organisms) { // For each organism (value) in
Organisms (array)
130.                // The radius to set the organisms on screen
131.                organismRadius = Math.min(20, Math.max(5, (int) (organism.getStats().getSizes()
* 10))); // Max radius of 20 and minimum of 5
132.
133.                if (organism.getStats().isDead()) { // If the organism is dead
134.                    if (showDeadOrganisms) { // and if user wants to show dead organisms then
paint them
135.                        g.setColor(Color.BLACK);
136.                        g.fillOval(organism.getxCoordinate(), organism.getyCoordinate(),
organismRadius, organismRadius); // Paint a red oval at position x, y, dimension 5, 5
137.                    }
138.                } else {
139.                    // Set color based on generation
140.                    int generation = organism.getStats().getGeneration();
141.                    if (generation < 10) g.setColor(Color.YELLOW); // Color for generation 0-
10
142.                    else if (generation < 30) g.setColor(Color.ORANGE); // Color for
generation 10-30

```

```

143.             else g.setColor(Color.RED); // Color for generation 30+
144.
145.             // If organism is asexual, paint as pink
146.             if (organism.getSex() == 2) g.setColor(Color.PINK);
147.
148.             g.fillOval(organism.getxCoordinate(), organism.getyCoordinate(),
organismRadius, organismRadius); // Paint a red oval at position x, y, dimension 5, 5
149.
150.             g2d.setColor(Color.BLACK); // Default outline colour
151.
152.             // If the organism is a predator, outline in red
153.             if (organism.isPredator()) g2d.setColor(Color.RED);
154.
155.             // If the organism is the selected one outline in cyan
156.             if(organism.getOrganismId() == GameWindow.organismSelected)
g2d.setColor(Color.CYAN);
157.
158.             // Draw the organism itself at the coordinates with the set radius
159.             g2d.drawOval(organism.getxCoordinate(), organism.getyCoordinate(),
organismRadius, organismRadius);
160.         }
161.     }
162. } else GameWindow.timeFactor += 500; // Allows the program to catch up with itself if
the process is expensive (file has not saved in time)
163.
164. }

```

C1.9 PerlinNoise.java

```

1. // Perlin noise class
2. // This class is an adaptation of an open source noise generator found on GitHub
3. // https://gist.github.com/alksily/7a85a1898e65c936f861ee93516e397d/
4. // The majority of this class was not coded by myself, I do not take credit for this code.
5. // I have added comments in places to better understand its function, however for references
it is better to check
6. // my documentation (3.1.1.1) or the documentation from the GitHub link above
7. public class PerlinNoise {
8.     private static double seed; // To store the seed value
9.     private long default_size; // The size for the noise function
10.    private int[] p; // Permutation array
11.
12.    // Constructor with a set seed
13.    public PerlinNoise(double newSeed) {
14.        seed = newSeed;
15.        this.init();
16.    }
17.
18.    // Constructor without a set seed
19.    public PerlinNoise() {
20.        this.init();
21.    }
22.
23.    // Initialisation method to set up the permutation array
24.    private void init() {
25.        this.p = new int[512];

```

```

26.     int[] permutation = new int[]{151, 160, 137, 91, 90, 15, 131, 13, 201, 95, 96, 53, 194,
233, 7, 225, 140, 36, 103, 30, 69, 142, 8, 99, 37, 240, 21, 10, 23, 190, 6, 148, 247, 120, 234, 75,
0, 26, 197, 62, 94, 252, 219, 203, 117, 35, 11, 32, 57, 177, 33, 88, 237, 149, 56, 87, 174, 20,
125, 136, 171, 168, 68, 175, 74, 165, 71, 134, 139, 48, 27, 166, 77, 146, 158, 231, 83, 111, 229,
122, 60, 211, 133, 230, 220, 105, 92, 41, 55, 46, 245, 40, 244, 102, 143, 54, 65, 25, 63, 161, 1,
216, 80, 73, 209, 76, 132, 187, 208, 89, 18, 169, 200, 196, 135, 130, 116, 188, 159, 86, 164, 100,
109, 198, 173, 186, 3, 64, 52, 217, 226, 250, 124, 123, 5, 202, 38, 147, 118, 126, 255, 82, 85,
212, 207, 206, 59, 227, 47, 16, 58, 17, 182, 189, 28, 42, 223, 183, 170, 213, 119, 248, 152, 2, 44,
154, 163, 70, 221, 153, 101, 155, 167, 43, 172, 9, 129, 22, 39, 253, 19, 98, 108, 110, 79, 113,
224, 232, 178, 185, 112, 104, 218, 246, 97, 228, 251, 34, 242, 193, 238, 210, 144, 12, 191, 179,
162, 241, 81, 51, 145, 235, 249, 14, 239, 107, 49, 192, 214, 31, 181, 199, 106, 157, 184, 84, 204,
176, 115, 121, 50, 45, 127, 4, 150, 254, 138, 236, 205, 93, 222, 114, 67, 29, 24, 72, 243, 141,
128, 195, 78, 66, 215, 61, 156, 180};
27.     this.default_size = 35L;
28.
29.     // Duplicate the permutation to create a larger array
30.     for (int i = 0; i < 256; ++i) {
31.         this.p[256 + i] = this.p[i] = permutation[i];
32.     }
33.
34. }
35.
36.     // Getter and Setter for the seed
37.     public static void setSeed(double newSeed) {
38.         seed = newSeed;
39.     }
40.
41.     public static double getSeed() {
42.         return seed;
43.     }
44.
45.     // Noise function for 2D coordinates.
46.     // EvSim uses a 2d map, therefore only needs 2d noise
47.     public double noise(double x, double y) {
48.         double value = 0.0D;
49.         double size = (double) this.default_size;
50.
51.         double initialSize;
52.         for (initialSize = size; size >= 1.0D; size /= 2.0D) {
53.             value += this.smoothNoise(x / size, y / size, 0.0D / size) * size;
54.         }
55.
56.         // Normalise against the initial size
57.         return value / initialSize;
58.     }
59.
60.     // Function to generate smooth noise at a given 3D point (x, y, z)
61.     public double smoothNoise(double x, double y, double z) {
62.         // Apply the seed to input coordinates
63.         x += seed;
64.         y += seed;
65.         z += seed;
66.
67.         // Calculate integer coordinates of the 3D point
68.         int X = (int) Math.floor(x) & 255;
69.         int Y = (int) Math.floor(y) & 255;
70.         int Z = (int) Math.floor(z) & 255;
71.
72.         // Calculate fractional part of the coordinates
73.         x -= Math.floor(x);
74.         y -= Math.floor(y);
75.         z -= Math.floor(z);
76.
77.         // Apply fade function to the fractional parts
78.         double u = this.fade(x);
79.         double v = this.fade(y);
80.         double w = this.fade(z);
81.
82.         // Hash coordinates to get gradient vectors
83.         int A = this.p[X] + Y;
84.         int AA = this.p[A] + Z;

```

```

85.     int AB = this.p[A + 1] + Z;
86.     int B = this.p[X + 1] + Y;
87.     int BA = this.p[B] + Z;
88.     int BB = this.p[B + 1] + Z;
89.
90.     // Perform tri-linear interpolation to get the final result
91.     return this.lerp(w, this.lerp(v, this.lerp(u, this.grad(this.p[AA], x, y, z),
this.grad(this.p[BA], x - 1.0D, y, z)),
92.                         this.lerp(u, this.grad(this.p[AB], x, y - 1.0D, z),
this.grad(this.p[BB], x - 1.0D, y - 1.0D, z))),
93.                         this.lerp(v, this.lerp(u, this.grad(this.p[AA + 1], x, y, z - 1.0D),
this.grad(this.p[BA + 1], x - 1.0D, y, z - 1.0D)),
94.                         this.lerp(u, this.grad(this.p[AB + 1], x, y - 1.0D, z - 1.0D),
this.grad(this.p[BB + 1], x - 1.0D, y - 1.0D, z - 1.0D))));}
95. }
96.
97. // Fade function for smoothing interpolation
98. private double fade(double t) {
99.     return t * t * t * (t * (t * 6.0D - 15.0D) + 10.0D);
100. }
101.
102. // Linear interpolation function
103. private double lerp(double t, double a, double b) {
104.     return a + t * (b - a);
105. }
106.
107. // Gradient function to compute dot product of gradient vectors and input coordinates
108. private double grad(int hash, double x, double y, double z) {
109.     int h = hash & 15;
110.     double u = h < 8 ? x : y;
111.     double v = h < 4 ? y : (h != 12 && h != 14 ? z : x);
112.     return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v);
113. }
114. }

```

C1.10 Organisms.java

```

1. import FoodTypes.*;
2. import Terrains.*;
3. import com.fasterxml.jackson.databind.ObjectMapper;
4. import org.json.simple.JSONArray;
5. import org.json.simple.JSONObject;
6. import org.json.simple.parser.JSONParser;
7. import org.json.simple.parser.ParseException;
8. import java.awt.*;
9. import java.io.File;
10. import java.io.FileReader;
11. import java.io.FileWriter;
12. import java.io.IOException;
13. import java.text.DecimalFormat;
14. import java.util.*;
15. import java.util.List;
16.
17. // This class updates all organisms statistics
18. // Contains many methods and functions, all performing different functions to update the
organisms statistics

```

```

19. // where applicable
20. public class Organisms {
21.     private String currentTerrain; // Global variables which have to be used commonly
throughout the class
22.     private int organismID; // Such as the current organism's id
23.     private int currentX; // Or coordinates
24.     private int currentY;
25.     private final Random random = new Random();
26.     private Stats stats = new Stats(); // The organisms stats
27.     private final Organism[] organisms = GameWindow.organisms; // Takes the organisms stored
in game window to be used here
28.     private final DecimalFormat decimalFormat = new DecimalFormat("#.##"); // Decimal
formatter used to smooth new values
29.     private double organismSpeed;
30.     private final List<JSONObject> organismsToAdd = new ArrayList<>();
31.     private int organismCount = organisms.length;
32.     public static boolean predatorPreyEnabled = true; // Boolean to define what to update as
per user requests
33.     public static boolean seasonsEnabled;
34.     public static String season;
35.
36.     // The constructor for Organisms, which calls and links together all methods and functions
37.     // The organisms array is looped through and each organism is updated if it is not dead
38.     public Organisms() {
39.         if (seasonsEnabled) updateSeason(); // Updates the season at the start of the year /
simulation if enabled
40.
41.         ObjectMapper objectMapper = new ObjectMapper();
42.         try { // To catch errors
43.             for (Organism organism : organisms) { // Loops through all organisms
44.                 if (!organism.getStats().isDead()) { // If the organism is dead, no need to
update
45.                     // Set the current attributes for the organism
46.                     organismID = organism.getOrganismId();
47.                     currentTerrain = organism.getCurrentTerrain();
48.                     currentX = organism.getxCoordinate();
49.                     currentY = organism.getyCoordinate();
50.                     stats = organism.getStats();
51.
52.                     // Calling the methods which update the organisms stats and position
53.                     makeDecision();
54.                     updateStats();
55.                     organismSpeed = stats.getSpeed(); // Speed updated here as it should use
the new stat
56.                     makeDecision();
57.                     organism.setCurrentTerrain(GenerateMap.getTerrain(currentX, currentY)); // Saves the new information from makeDecision()
58.                     organism.setxCoordinate(currentX);
59.                     organism.setyCoordinate(currentY);
60.                     organism.setStats(stats); // Saves the new stats from updateStats()
61.                 }
62.             }
63.             objectMapper.writeValue(new File("organisms.json"), organisms); // Syncs the JSON
with what is stored in organisms
64.             if (!organismsToAdd.isEmpty())
65.                 addNewOrganisms(organismsToAdd); // If there are new organisms to add, the
JSON is updated with the new organisms through addNewOrganisms()
66.         } catch (Exception e) {
67.             throw new RuntimeException(e); // Catch and print any errors
68.         }
69.     }
70.
71.     // This method checks the current season and increments if due to.
72.     // If the user has chosen to turn off seasons then this method is not called.
73.     private void updateSeason() {
74.         if (GameWindow.years == 0 || season == null) { // If it is the beginning, select a
random season
75.             int randomSeason = random.nextInt(0, 4);
76.             season = switch (randomSeason) { // Random season
77.                 case 0 -> "Winter",

```

```

78.             case 1 -> "Spring";
79.             case 2 -> "Summer";
80.             default -> "Autumn";
81.         );
82.     }
83.     if (GameWindow.years % 4 == 0 && GameWindow.years != 0) { // Update the season every 4 years
84.         season = switch (season) { // Move onto the next season
85.             case "Winter" -> "Spring";
86.             case "Spring" -> "Summer";
87.             case "Summer" -> "Autumn";
88.             default -> "Winter";
89.         };
90.     }
91. }
92.
93. // The calculated decision made by each organism to move to the best position based on survivability
94. // is affected by organism proximity and terrain type
95. private void makeDecision() {
96.     // Lists which will contain the x and y coordinates which can be moved to
97.     List<Point> positions = new ArrayList<>();
98.
99.     // Populates the list created above with the appropriate coordinates
100.    for (int xCounter = (int) (0 - organismSpeed); xCounter < organismSpeed; xCounter++) {
101.        int newX = Math.abs(currentX + xCounter);
102.        for (int yCounter = (int) (0 - organismSpeed); yCounter < organismSpeed;
yCounter++) { // Check each y and x coordinate around
103.            int newY = Math.abs(currentY + yCounter);
104.            positions.add(new Point(newX, newY)); // Add it to the possible positions
105.        }
106.    }
107.
108.    // This ensures that the new X and Y coordinates are not outside the bounds of the window
109.    for (int index = positions.size() - 1; index >= 0; index--) {
110.        Point position = positions.get(index);
111.        if (position.x >= GameWindow.WIDTH - 300 || position.y >= GameWindow.HEIGHT - 45)
{
112.            positions.remove(index); // Removes it if so
113.        }
114.    }
115.
116.    Organism organismToSeek = organismToSeek(); // Find the organism to seek
117.
118.    // List of survivability levels for each x and y position
119.    List<Double> listOfSurvivability = new ArrayList<>();
120.    for (Point position : positions) listOfSurvivability.add(getSurvivability(position.x,
position.y, organismToSeek));
121.
122.    // Find position with one of the highest survivability
123.    double maxSurvivability =
listOfSurvivability.stream().mapToDouble(Double::doubleValue).max().orElse(0); // Find the highest survivability value in the list using stream
124.
125.    // List to track all the highest ones
126.    List<Integer> highestSurvivabilityPositions = new ArrayList<>();
127.
128.    // If the survivability is equal to the highest survivability, add the index to the list of ideal positions
129.    for (int counter3 = 0; counter3 < listOfSurvivability.size(); counter3++)
130.        if (listOfSurvivability.get(counter3) == maxSurvivability)
highestSurvivabilityPositions.add(counter3);
131.
132.    // Choose a random position of the list, this translates to the position of the x and y coordinates
133.    int randomValue =
highestSurvivabilityPositions.get(random.nextInt(highestSurvivabilityPositions.size()));
134.    currentX = positions.get(randomValue).x; // The x and y coordinates of the chosen position are now set and updated
135.    currentY = positions.get(randomValue).y;

```

```

136.         // This ensures that the new X and Y coordinates are not outside the bounds of the
137.         window
138.         // The x and y coordinate are compared against the width and height of the gameWindow,
if higher or lower,
139.         // the coordinates are adjusted in the appropriate direction
140.         // This was checked earlier, this is in position as a final check - ensures no errors
141.         if (currentX >= GameWindow.WIDTH - 260) {
142.             currentX = GameWindow.WIDTH - 300;
143.         } else if (currentX < 0) currentX = 10;
144.         if (currentY >= GameWindow.HEIGHT - 35) {
145.             currentY = GameWindow.HEIGHT - 75;
146.         } else if (currentY < 0) currentY = 10;
147.     }
148.
149.     // Gets the survivability of an organism at the coordinate position newX, newY
150.     // Survivability depends on organism proximity and terrain type
151.     public double getSurvivability(int newX, int newY, Organism organismToSeek) {
152.         double survivability = 0.1; // Default value
153.         double chanceToMisjudge = random.nextDouble(0, 1); // Small chance to make a mistake
154.         String newTerrain = GenerateMap.getTerrain(newX, newY); // Gets the terrain at
position newX, newY
155.
156.         if (chanceToMisjudge <= 0.02) { // Small chance to misjudge (move to a position it
doesn't particularly benefit from)
157.             if (organismToSeek != null) { // If it is not null (no organisms left to seek)
158.                 // Calculate the Euclidean distance between current organism and target
organism
159.                 double currentDistance = Math.sqrt(Math.pow(organismToSeek.getxCoordinate() -
newX, 2) + Math.pow(organismToSeek.getyCoordinate() - newY, 2));
160.
161.                 // Calculates a value between -2 and 2 to distance how close the organism is
to the target organism.
162.                 // The closer the value is, the higher the survivability will be set
163.                 survivability = (1.0 / (1.0 + currentDistance)) * 10;
164.                 survivability =
Double.parseDouble(decimalFormat.format(Math.max(Math.min(survivability - 1, 1.5), 0))); // Limit
it to 0 and 1.5
165.             }
166.
167.             // Terrain survivability calculations
168.             if
(newTerrain.equals(organisms[this.organismID].getStats().getPreferredTerrain()))
169.                 survivability = survivability + 0.15; // If it is the preferred terrain
receive a boost
170.             else if
(newTerrain.equals(organisms[this.organismID].getStats().getSecondPreferredTerrain()))
171.                 survivability = survivability + 0.10; // If it is the second most preferred
terrain receive a boost
172.             else if
(newTerrain.equals(organisms[this.organismID].getStats().getThirdPreferredTerrain()))
173.                 survivability = survivability + 0.05; // If it is the third most preferred
terrain receive a boost
174.             } else survivability = 1.5; // Default to 1.5 as a misjudgement
175.
176.             return survivability; // Returns the calculated survivability
177.         }
178.
179.         // This will cover the organisms selection of organism to seek out, whether it be as prey
or as breeding
180.         public Organism organismToSeek() {
181.             Organism currentOrganism = organisms[this.organismID]; // Localises the current
organism
182.             List<Organism> closestOrganisms = listClosestOrganisms(currentOrganism); // Creates
the list of closest organisms
183.             Organism targetOrganism = new Organism(); // Will be set to the target organism when
found
184.
185.             // If the organism is a predator and cannot breed
186.             if (currentOrganism.isPredator() && !(stats.getAge() >= 15) &&
!(stats.getAgeLastBred() < stats.getAge() - 5)) {

```

```

187.         for (Organism organism : closestOrganisms) {
188.             if (!organism.isPredator()) {
189.                 targetOrganism = organism; // Target the closest organism as prey
190.                 break;
191.             }
192.         }
193.     }
194.     // If the organism is not a predator and cannot breed
195.     else if (!currentOrganism.isPredator() && !(stats.getAge() >= 15) &&
196.             !(stats.getAgeLastBred() < stats.getAge() - 5)) {
197.         targetOrganism = null; // Do not target for another organism
198.     }
199.     // If the organism is not a predator but can breed
200.     else if (!currentOrganism.isPredator() && stats.getAge() >= 15 &&
201.             stats.getAgeLastBred() < stats.getAge() - 5) {
202.         for (Organism organism : closestOrganisms) {
203.             // If the other organism is not a predator, not the same sex, and not asexual
204.             if (!organism.isPredator() && organism.getStats().getAge() >= 15 &&
205.                 organism.getStats().getAgeLastBred() < organism.getStats().getAge() - 5 &&
206.                 organism.getSex() != 2 && organism.getSex() !=
207.                 currentOrganism.getSex()) {
208.                 targetOrganism = organism; // Sets to the closest organism it can breed
209.                 with
210.                     break;
211.                 }
212.             }
213.         return targetOrganism; // Return the target organism
214.     }
215.
216.     // This returns a list of closest organisms to organism sent in
217.     // Returns a list of points which contains an x and y coordinate
218.     private List<Organism> listClosestOrganisms(Organism currentOrganism) {
219.         List<Organism> closestOrganisms = new ArrayList<>();
220.         Point currentLocation = new Point(currentOrganism.getxCoordinate(),
221.                                         currentOrganism.getyCoordinate());
222.
223.         // Filter and sort Organisms to Points which are closest to the current organism
224.         // Uses .stream to process the array
225.         // Filters out organisms that have the same ID and are dead
226.         // Maps each organism to a Point using the organisms x and y coordinates
227.         // Sorts the points created and converts into a list
228.         List<Point> organismPoints = Arrays.stream(organisms)
229.             .filter(organism -> organism.getOrganismId() !=
230.                     currentOrganism.getOrganismId() && !organism.getStats().isDead())
231.             .map(organism -> new Point(organism.getxCoordinate(),
232.                                         organism.getyCoordinate())).sorted(Comparator.comparingDouble(p ->
233.                                         Math.sqrt(Math.pow(currentLocation.x - p.x, 2) + Math.pow(currentLocation.y - p.y, 2))).toList();
234.
235.         // Transfers the list of points into a list of correlating organisms
236.         for (Point point : organismPoints) {
237.             for (Organism organism : organisms) {
238.                 if (organism.getxCoordinate() == point.x && organism.getyCoordinate() ==
239.                     point.y && !organism.getStats().isDead()) {
240.                         closestOrganisms.add(organism);
241.                         break;
242.                     }
243.                 }
244.             }
245.         return closestOrganisms; // Returns the list of closest organisms
246.     }
247.     // This method calls together all update methods such as updateSpeed and updateAwareness
248.     // Only updates if an organism isn't dead (see constructor).
249.     private void updateStats() {
250.         // Fetch the current terrain to be used for some methods

```

```

248.     TerrainAttributes terrainAttributes = getTerrainAttributes(currentTerrain);
249.
250.     updateAwareness(terrainAttributes); // Updating the organism's awareness
251.     updateFood(terrainAttributes); // Updating the food an organism eats
252.     stats.setSizeFactor(stats.getSizeFactor() + stats.getEatenFood().foodFactor); // 
Update the total food factor
253.     updateSpeed(); // Updating the speed of an organism
254.     updateTemperature(terrainAttributes); // Updating the current temperature factor
255.     updateHabitat(terrainAttributes); // Updating the preferred habitat of an organism if
applicable
256.     updateHealth(); // Updating the health of an organism
257.     if (organisms[this.organismID].getSex() != 2)
        checkAsexual(); // If an organism is not already asexual, check if it can become
one
258.     if (organisms[this.organismID].getStats().getTSB() == null)
        checkTSB(); // Checks if the organism should have TSB, if it doesn't already
259.     if (!organisms[this.organismID].isPredator() && predatorPreyEnabled)
        checkPredation(); // Checks if the organism should be a predator
260.     if (organisms[this.organismID].getSex() == 2 && stats.getAgeLastBred() <
stats.getAge() - 40 && stats.getAge() >= 15)
        // If the organism is asexual give it opportunity to create a child
261.         createChildOrganism(organisms[this.organismID], organisms[this.organismID]);
262.
263.         stats.setAge(stats.getAge() + 1); // adding 1 year to the organisms age
264.
265.     }
266.
267. }
268.
269.
270. // This method returns the current terrain as an object
271. // So it can be used by certain methods in this class
272. private TerrainAttributes getTerrainAttributes(String currentTerrain) {
273.     TerrainAttributes terrainAttributes; // The object which is returned
274.
275.     terrainAttributes = switch (currentTerrain) {
276.         case "Grass" -> new Grass(); // Returns the grass class from Terrains package
277.         case "Water" -> new Water(); // Returns the water class from Terrains package
278.         case "Mountain" -> new Mountain();
279.         case "Beach" -> new Beach();
280.         case "Snow" -> new Snow();
281.         case "Forest" -> new Forest();
282.         case "Deep Water" -> new DeepWater();
283.         default -> new TerrainAttributes();
284.     };
285.     return terrainAttributes; // Returns the terrain as an object
286. }
287.
288. // This is similar to the getTerrainAttributes
289. // Returns the food type as an object
290. // Takes in a string (how food is stored) and returns the linking food type
291. private FoodTypes getFoodType(String foodName) {
292.     FoodTypes foodType; // The object which is returned
293.
294.     foodType = switch (foodName) { // Switch statement to find and set the food type
295.         case "Grass Seeds" -> new GrassSeeds(); // Returns the grass class from Terrains
package
296.         case "Beach Grass Seeds" -> new BeachGrassSeeds(); // Returns the water class from
Terrains package
297.         case "Grasses and Herbs" -> new GrassesHerbs(); // ""
298.         case "Fungi" -> new Fungi();
299.         case "Leaves / Large Plants" -> new LeavesPlants();
300.         case "Tree Roots / Plant Remains" -> new RootsPlantRemains();
301.         case "Berries" -> new Berries();
302.         case "Coastal Fruit / Plants" -> new CoastalFruitPlants();
303.         case "Shrub Berries" -> new ShrubBerries();
304.         case "Shrubs" -> new Shrubs();
305.         case "Coniferous Tree Spawn" -> new ConiferousTreeSpawn();
306.         case "Tree Spawn" -> new TreeSpawn();
307.         case "Moss / Lichen" -> new MossLichen();
308.         case "Aquatic Plants" -> new AquaticPlants();
309.         case "Algae" -> new Algae();
310.         case "Insects and Insect Larvae" -> new InsectsLarvae();
311.         case "Shellfish and Crustaceans" -> new ShellfishCrustaceans();
312.         case "Plankton" -> new Plankton();

```

```

313.             case "Deep Sea Fish" -> new DeepSeaFish();
314.             case "Large Deep Sea Fish" -> new LargeDeepSeaFish();
315.             default -> new FoodTypes();
316.         };
317.         return foodType; // Return the food type
318.     }
319.
320.     // This method updates an organisms awareness, taking into account the terrain's
321.     // visibility
322.     private void updateAwareness(TerrainAttributes terrainAttributes) {
323.         // Calls the method to check if the organism has encountered another organism
324.         checkOrganismProximity(stats.getCurrentAwareness());
325.         // Sets the variables that are read throughout
326.         double terrainVisibility = terrainAttributes.visibility; // The visibility attribute
327.         for each terrain
328.             double organismsEncountered = stats.getOrganismsEncountered(); // The updated number
329.             of organisms encountered
330.
331.             List<Double> awarenessHistory = stats.getAwarenessHistory(); // The list of previous
332.             organism awareness'
333.             int awarenessHistoryArraySize; // Required as the size of the array is not always >= 4
334.             // This switch expression sets awarenessHistoryArraySize and totalHistory variables.
335.             // Total history is a total of previous awareness', it is used to get the average
336.             // awareness (part of the algorithm)
337.             // awarenessHistoryArraySize is dependent on how old the organism is, therefore needs
338.             // to also be set depending on
339.             // the size of the array stored in stats.
340.             double totalHistory = switch (awarenessHistory.size()) {
341.                 case 0 -> {
342.                     awarenessHistoryArraySize = 1;
343.                     yield 0.0;
344.                 }
345.                 case 1 -> {
346.                     awarenessHistoryArraySize = 2;
347.                     yield awarenessHistory.get(0);
348.                 }
349.                 case 2 -> {
350.                     awarenessHistoryArraySize = 3;
351.                     yield awarenessHistory.get(0) + awarenessHistory.get(1);
352.                 }
353.                 case 3 -> {
354.                     awarenessHistoryArraySize = 4;
355.                     yield awarenessHistory.get(0) + awarenessHistory.get(1) +
356.                         awarenessHistory.get(2);
357.                 }
358.                 default -> {
359.                     awarenessHistoryArraySize = 5;
360.                     yield awarenessHistory.get(0) + awarenessHistory.get(1) +
361.                         awarenessHistory.get(2) + awarenessHistory.get(3);
362.                 }
363.             };
364.
365.             // Calculates the awareness, which is set to 2 decimal places (to ensure no long
366.             // decimals)
367.             // and used to calculate the new currentAwareness
368.             double awareness = Double.parseDouble(decimalFormat.format(terrainVisibility +
369.                 organismsEncountered));
370.
371.             // Calculates the organisms new currentAwareness. Set to 2 decimal places (to ensure
372.             // no long decimals)
373.             double currentAwareness = Math.max(0.4, (awareness + totalHistory) /
374.                 awarenessHistoryArraySize) + organismsEncountered;
375.
376.             // Both use same calculations as logic in documentation (see 3.3.1.1)
377.
378.             if (Objects.equals(stats.getTSB(), organisms[organismID].getCurrentTerrain()))
379.                 currentAwareness = currentAwareness + 0.2; // If it has a TSB, give an awareness
380.                 boost
381.
382.             stats.setCurrentAwareness(Double.parseDouble(decimalFormat.format(currentAwareness)));
383.
384.             // Updates the local stats variable
385.             awarenessHistory.add(0, awareness); // Adds the awareness to the locally held

```

```

370.         awarenessHistory = awarenessHistory.subList(0, Math.min(awarenessHistory.size(), 4));
// Restricts the size of awarenessHistory to 4
371.         stats.setAwarenessHistory(awarenessHistory); // Updates the awarenessHistory which is
held in JSON
372.     }
373.
374.     // This method updates the number of organism encounters which are stored locally.
375.     // Takes the double doubleAwareness as an input, which is the area in which an organism
can see (awareness)
376.     private void checkOrganismProximity(double doubleAwareness) {
377.         // Multiplies by 10 so that the bounds are 10px instead of 1
378.         int currentAwareness = (int) (doubleAwareness * 10); // Passing the awareness into an
integer, so it can be compared to coordinates.
379.         for (Organism organism : organisms) {
380.             if (organism.getOrganismID() != this.organismID && !organism.getStats().isDead())
{ // So it does not compare to its own (would always be true)
381.                 int checkX = organism.getxCoordinate(), checkY = organism.getyCoordinate(); // The
coordinates to check
382.                 int xDifference = Math.abs(this.currentX - checkX); // Returning the
difference between mouse and organism coordinates
383.                 int yDifference = Math.abs(this.currentY - checkY); // Returns as an absolute
number, so it can be compared to only one bound
384.                 // If the coordinates are within the bound, then increase the number of
organisms encountered
385.                 if (xDifference <= currentAwareness && yDifference <= currentAwareness) { // Bound
is the organisms awareness
386.                     stats.setOrganismsEncountered(stats.getOrganismsEncountered() + 0.1); // Increase
the number of organisms encountered
387.                     newOrganismEncounter(this.organismID, organism.getOrganismID());
388.                     if (organism.getSex() != organisms[this.organismID].getSex() &&
organism.getStats().getAge() >= 15 && stats.getAge() >= 15 && organism.getStats().getAgeLastBred() <
organism.getStats().getAge() - 5 && organisms[this.organismID].getStats().getAgeLastBred() <
organisms[this.organismID].getStats().getAge() - 5)
389.                         stats.setOrganismsMet(stats.getOrganismsMet() + 1);
390.                     stats.setTotalOrganismsMet(stats.getTotalOrganismsMet() + 1);
391.                 }
392.             }
393.         }
394.         stats.setOrganismsEncountered(Math.min(0.5, stats.getOrganismsEncountered())); // Hard-cap the
amount of organisms encountered
395.         stats.setOrganismsEncountered(Math.max(0.0, stats.getOrganismsEncountered())); // Ensure it's not
negative
396.     }
397.
398.     // This method updates the organisms current stored food factor, as well as preferred and
eaten food
399.     private void updateFood(TerrainAttributes terrainAttributes) {
400.         // Finding the food types for the current terrain
401.         ArrayList<String> availableFoods = terrainAttributes.foods;
402.
403.         // This is in the case that a file is being loaded
404.         // The preferred food type of organisms was stored as a string
405.         // Therefore if it is currently holding a string (and not null) then transfer it over
406.         if (stats.getPreferredFoodString() != null) {
407.             stats.setPreferredFood(getFoodType(stats.getPreferredFoodString())); // Transfer
and store the food type
408.             stats.setPreferredFoodString(null); // Return the preferred string to be null
409.         }
410.
411.         if (stats.getAge() < 5 && stats.getPreferredFood() == null) {
412.             // If the organism is younger than 5, as by documentation the organism can eat
anything
413.             // What the organism eats within the first 5 years is added to a list of recent
foods
414.             // Once the organism passes year 5, they eat the food they ate the most within the
first 5 years
415.             String tempFood; // The food that will be chosen
416.             FoodTypes potentialFood1 = getFoodType(availableFoods.get(0)), potentialFood2 =
getFoodType(availableFoods.get(1)), potentialFood3 = getFoodType(availableFoods.get(2)); // The
foods that can be chosen
417.

```

```

418.         double totalAvailability = potentialFood1.foodAvailability +
potentialFood2.foodAvailability + potentialFood3.foodAvailability;
419.         double randomChoice = random.nextDouble(0, totalAvailability);
420.         if (randomChoice > 0 && randomChoice <= potentialFood1.foodAvailability)
421.             tempFood = availableFoods.remove(0); // Returns the first item from the list
422.         else if (randomChoice > potentialFood1.foodAvailability && randomChoice <=
potentialFood2.foodAvailability + potentialFood1.foodAvailability)
423.             tempFood = availableFoods.remove(1); // Returns the second item from the list
424.         else tempFood = availableFoods.remove(2); // Returns the last food in the list
425.
426.         List<FoodTypes> recentFoods = new ArrayList<>();
427.
428.         if (stats.getRecentFood() != null) recentFoods = stats.getRecentFood();
429.         recentFoods.add(getFoodType(tempFood)); // Adds the most recent food
430.         stats.setRecentFood(recentFoods); // Updates the food eaten list
431.         stats.setEatenFood(getFoodType(tempFood)); // The food which was selected is set
to as eaten
432.     } else if (stats.getAge() == 5 && stats.getPreferredFood() == null) {
433.         // Once the organism passes year 5, they select preferred food based on the most
common food found during first 5 years
434.         List<FoodTypes> recentFoods = stats.getRecentFood();
435.         int count = 0;
436.         FoodTypes mostCommon = new FoodTypes();
437.         // Choosing the most common food that was eaten
438.         for (int counter1 = 0; counter1 < recentFoods.size(); counter1++) {
439.             int counter3 = 0;
440.             for (FoodTypes recentFood : recentFoods) if (recentFoods.get(counter1) ==
recentFood) counter3++;
441.             if (counter3 > count) {
442.                 count = counter3;
443.                 mostCommon = recentFoods.get(counter1);
444.             }
445.         }
446.         stats.setPreferredFood(mostCommon); // Set the preferred food to the food type of
most eaten food
447.         stats.setEatenFood(mostCommon); // Set the eaten food to the food type of most
eaten food
448.         stats.setRecentFood(new ArrayList<>()); // Clear the array to free space
449.     } else {
450.         // If the age is over 5, get the preferred food. Check if the preferred food is
part of the available foods list,
451.         // if it is set to the eaten food, if it isn't, use foodAvailability to set a new
eaten food from then environment
452.         // preferred and eaten food below
453.         if (availableFoods.contains(stats.getPreferredFood().foodName))
454.             stats.setEatenFood(stats.getPreferredFood()); // If the preferred food is
available, then eat it
455.         else {
456.             // If preferred food is not available, check each food available for the
closest match to the preferred food
457.             for (String food : availableFoods) {
458.                 FoodTypes currentFood = getFoodType(food);
459.                 if (currentFood.parentName.equals(stats.getPreferredFood().parentName)) {
460.                     stats.setEatenFood(currentFood); // Within the same food category,
therefore closest
461.                     break;
462.                 } else if (currentFood.landBasedFood ==
stats.getPreferredFood().landBasedFood) {
463.                     stats.setEatenFood(currentFood); // Within same food type, continue
checking as may not be same class
464.                     if (food.equals(availableFoods.get(availableFoods.size() - 1)))
465.                         break; // If there are no more available foods, then this is the
closest
466.                 } else
467.                     stats.setEatenFood(getFoodType(food)); // If all available food types
are not within the same food type or category
468.             }
469.         }
470.     }
471.     // If it has a preferred food then update depending on that
472.     if (stats.getPreferredFood() != null) {

```

```

473.         FoodTypes preferredFood = stats.getPreferredFood(); // Preferred Food
474.         FoodTypes eatenFood = stats.getEatenFood(); // Current Food
475.         double foodFactor;
476.
477.         // If the two foods are the same
478.         if (Objects.equals(preferredFood.foodName, eatenFood.foodName)) foodFactor =
eatenFood.foodFactor;
479.         else if (Objects.equals(preferredFood.parentName, eatenFood.parentName))
foodFactor = eatenFood.foodFactor * 0.8; // If they are part of the same food
class
481.         else if ((preferredFood.waterBasedFood == eatenFood.waterBasedFood) ||
(preferredFood.landBasedFood && eatenFood.landBasedFood))
482.             foodFactor = eatenFood.foodFactor * 0.6; // If they are the same food type
483.         else foodFactor = eatenFood.foodFactor * 0.3; // If they are different food types
484.
485.         if (Objects.equals(organisms[organismID].getCurrentTerrain(), stats.getTSB()))
foodFactor = foodFactor + (foodFactor * 0.4); // If it has a TSB, then give a
food factor benefit
486.
487.         stats.setFoodFactor(foodFactor);
488.     } else {
489.         // If the organism is younger than 5 then just normal food factor
490.         stats.setFoodFactor(stats.getEatenFood().foodFactor);
491.     }
492. }
493.
494.
495. // Updates the organisms speed
496. // Takes the old speed, age and foodFactor into a formula
497. private void updateSpeed() {
498.     // newSpeed = oldSpeed * (1.01 + foodFactor * age / (1 + age ^ 3))
499.     double oldSpeed = stats.getSpeed();
500.     double foodFactor = stats.getFoodFactor();
501.     int age = stats.getAge();
502.
503.     // Formatted to 2dp to make more readable and take less space in JSON
504.     double newSpeed = Double.parseDouble(decimalFormat.format(oldSpeed * (1.01 +
foodFactor * age / (1 + Math.pow(age, 3)))));
505.     stats.setSpeed(newSpeed); // Updating the stats
506. }
507.
508. // This method updates the organisms preferred temperature as well as the temperature
factor
509. // Takes in the current temperature of the terrain, as well as the preferred temp (if
applicable)
510. // Is affected by the current season (if enabled) See 3.10 Documentation
511. private void updateTemperature(TerrainAttributes terrainAttributes) {
512.     int temperatureChange = 0;
513.     if (seasonsEnabled && season != null) {
514.         temperatureChange = switch (season) { // The temperature change which will take
place due to the season
515.             case "Winter" -> random.nextInt(-11, -7); // Bound is exclusive so set to -7
516.             case "Spring" -> random.nextInt(-8, -2);
517.             case "Summer" -> random.nextInt(8, 11);
518.             default -> random.nextInt(3, 9);
519.         };
520.     }
521.
522.     int newTemp = terrainAttributes.temperature + temperatureChange; // The new
temperature for the organism
523.
524.     if (stats.getAge() < 5) { // If the organism is younger than 5, it can handle any
temperature
525.         List<Integer> recentTemp = stats.getRecentTemp(); // Update the list of recent
temperatures
526.         recentTemp.add(newTemp); // Add in the new temperature
527.         stats.setRecentTemp(recentTemp); // Update the array in stats
528.     } else if (stats.getAge() == 5) { // If the organism is 5 years old, select a
preferred temperature
529.         List<Integer> recentTemp = stats.getRecentTemp(); // Gets the list of previous
temperatures
530.         int mostCommon;

```

```

531.          // Find the average temperature from the 5 previous years
532.          mostCommon = recentTemp.get(0) + recentTemp.get(1) + recentTemp.get(2) +
recentTemp.get(3) + recentTemp.get(4);
533.          mostCommon = mostCommon / 5;
534.          stats.setPreferredTemp(mostCommon); // Set that as the preferred temperature
535.          stats.setRecentTemp(new ArrayList<>()); // Clear the recent temperature array to
save space
536.          stats.setCurrentTemp(newTemp);
537.      } else { // Find the organisms temperature factor - based on the organisms preferred
and current temperature
538.          int preferredTemp = stats.getPreferredTemp();
539.          int difference = Math.abs(newTemp - preferredTemp); // Finding the difference (as
a positive (absolute value)) between the preferred and the current
540.          stats.setTempFactor(1 - difference * 0.04); // Setting the new temperature factor
which will be used later.
541.          stats.setCurrentTemp(newTemp);
542.      }
543.  }
544.
545.  // Compare the attributes of the current terrain to the organisms preferred attributes.
546.  // If the attributes are more favourable than its other preferred terrains, add it as
preferred to the appropriate position
547.  private void updateHabitat(TerrainAttributes terrainAttributes) {
548.      String preferredTerrain = stats.getPreferredTerrain();
549.      String secondPreferredTerrain = stats.getSecondPreferredTerrain();
550.      String thirdPreferredTerrain = stats.getThirdPreferredTerrain();
551.      if (stats.getAge() >= 5 && !Objects.equals(currentTerrain, preferredTerrain) &&
!Objects.equals(currentTerrain, secondPreferredTerrain) && !Objects.equals(currentTerrain,
thirdPreferredTerrain)) { // If the organism has chosen its preferred attributes
552.          int newCount = newCount(terrainAttributes); // The current terrain's score
553.
554.          int currentCount = newCount(getTerrainAttributes(preferredTerrain)); // The
preferred terrain's score
555.          int secondCurrentCount = newCount(getTerrainAttributes(secondPreferredTerrain));
// The score of the second and third terrains
556.          int thirdCurrentCount = newCount(getTerrainAttributes(thirdPreferredTerrain));
557.
558.          // Checks the current score compared to other preferred terrains
559.          if (currentCount < newCount) {
560.              stats.setPreferredTerrain(terrainAttributes.terrainId);
561.              stats.setSecondPreferredTerrain(preferredTerrain); // Moving the other items
down the list
562.              stats.setThirdPreferredTerrain(secondPreferredTerrain);
563.          } else if (secondCurrentCount < newCount &&
!preferredTerrain.equals(terrainAttributes.terrainId)) {
564.              stats.setSecondPreferredTerrain(terrainAttributes.terrainId);
565.              stats.setThirdPreferredTerrain(secondPreferredTerrain); // Moving the other
items down the list
566.          } else if (thirdCurrentCount < newCount &&
!preferredTerrain.equals(terrainAttributes.terrainId) &&
!secondPreferredTerrain.equals(terrainAttributes.terrainId))
567.              stats.setThirdPreferredTerrain(terrainAttributes.terrainId);
568.          } else stats.setPreferredTerrain(preferredTerrain); // Default to the current terrain
569.      }
570.
571.      // Adds up the score of the terrain sent in, is used to check against preferred terrains
572.      private int newCount(TerrainAttributes terrainAttributes) {
573.          if (!Objects.equals(terrainAttributes.terrainId, "Default")) { // If it is not set to
the TerrainAttributes (null attributes)
574.              int currentTemp = terrainAttributes.temperature;
575.              List<String> availableFoods = terrainAttributes.foods;
576.              int preferredTemp = stats.getPreferredTemp();
577.              String preferredFood = stats.getPreferredFood().foodName;
578.
579.              int newCount = 0;
580.              if (availableFoods.contains(preferredFood)) {
581.                  newCount = newCount + 2; // If the food is contained in that terrain get an
extra point
582.              } else newCount = newCount + 1;
583.
584.              int tempDiff = Math.abs(preferredTemp - currentTemp);

```

```

585.             if (tempDiff < 5)
586.                 newCount = newCount + 3; // If the preferred and current temperatures are
587.                 // within a range of 5 from each other get 3 points
588.             else if (tempDiff < 10) newCount = newCount + 2; // Within 10 then 2 points
589.             else newCount = newCount + 1;
590.         return newCount;
591.     }
592.     return 0; // Default the score at 0
593. }
594.
595. // Updates the health of an organism
596. // Uses the organisms temperature and food factors to calculate the new health
597. private void updateHealth() {
598.     if (!stats.isDead()) { // If the organism isn't dead already
599.         double currentHealth = stats.getHealth(); // The current health
600.         double healthDecrease = 0.5; // Default decrease
601.         double temperatureDecrease = Math.abs(1 - stats.getTempFactor()); // The decrease
due to temperature factor
602.         double foodDecrease = Math.abs(1 - stats.getFoodFactor()); // The decrease due to
food factor
603.         double totalDecrease = healthDecrease + temperatureDecrease + foodDecrease; // The
total decrease amount
604.
605.         if (Objects.equals(stats.getTSB(), organisms[organismID].getCurrentTerrain()))
606.             totalDecrease = totalDecrease + 0.2;
607.
608.         currentHealth = currentHealth - totalDecrease; // Decrease the health by the
decreasing amount
609.
610.         if (currentHealth <= 0) { // If the organism is dead, set it to be dead and then
round the health to 0
611.             stats.setDead(true);
612.             currentHealth = 0.0;
613.         } else if (currentHealth > 100)
614.             stats.setHealth(100.0); // If the health is ever greater than 100, cap it at
100
615.         stats.setHealth(Double.parseDouble(decimalFormat.format(currentHealth))); // /
Setting to 2 decimal places
616.     }
617. }
618.
619. // This will contain both breeding cycle calculations and predator / prey cycle
calculations
620. private void newOrganismEncounter(int originOrganismId, int secondaryOrganismId) {
621.     Organism originOrganism = organisms[originOrganismId]; // The two organisms are
fetched
622.     Organism secondaryOrganism = organisms[secondaryOrganismId];
623.     int defaultAgeBarrier = 5;
624.     if (organisms.length > 150) defaultAgeBarrier = 20; // The amount of time between
organisms giving birth
625.     else if (organisms.length > 100) defaultAgeBarrier = 10;
626.
627.     if (originOrganism.getSex() != secondaryOrganism.getSex() && 30 >
Math.abs(originOrganism.getStats().getAge() - secondaryOrganism.getStats().getAge())) { // If they
are male and female
628.         if (originOrganism.getStats().getAgeLastBred() <
originOrganism.getStats().getAge() - defaultAgeBarrier &&
secondaryOrganism.getStats().getAgeLastBred() < secondaryOrganism.getStats().getAge() -
defaultAgeBarrier) { // If both are older than 15
629.             if (Math.abs(originOrganism.getxCoordinate() -
secondaryOrganism.getxCoordinate()) <= 5 && Math.abs(originOrganism.getyCoordinate() -
secondaryOrganism.getyCoordinate()) <= 5) { // If they are within a radius of 5
630.                 if (originOrganism.getSex() == 1) { // If the current organism is male
631.                     createChildOrganism(originOrganism, secondaryOrganism); //
createChildOrganism (male, female)
632.                     originOrganism.getStats().setAgeLastBred(stats.getAge()); // Update
age last bred for both
633.                     secondaryOrganism.getStats().setAgeLastBred(stats.getAge());
634.                 } else if (originOrganism.getSex() == 0) { // If current organism is
female

```

```

635.          createChildOrganism(secondaryOrganism, originOrganism); //
636. createChildOrganism (male, female)
637.         secondaryOrganism.getStats().setAgeLastBred(stats.getAge()); // Update
age last bred for both
638.         }
639.     }
640.   }
641. } else if (originOrganism.isPredator() || secondaryOrganism.isPredator()) { // Cause a
fight if the organisms are the same sex and one is a predator
642.
643.   if (originOrganism.isPredator() && !secondaryOrganism.isPredator())
// If the origin organism is the predator and secondary is prey
644.     newOrganismAttack(originOrganism, secondaryOrganism);
645.
646.   else if (!originOrganism.isPredator() && secondaryOrganism.isPredator())
// If the origin organism is prey and secondary is predator
647.     newOrganismAttack(secondaryOrganism, originOrganism);
648.
649.   else {
// If both are predators, then a random one attacks with the other defending
650.     if (random.nextInt(0, 2) == 0) newOrganismAttack(originOrganism,
secondaryOrganism);
651.     else newOrganismAttack(secondaryOrganism, originOrganism);
652.   }
653. }
654. }
655. }
656. }
657. }
658.
659. // Creates a JSON Object (new child) which will be added to a list,
660. // which will be added to the JSON at the end of the year
661. @SuppressWarnings("unchecked") // As it can be ignored
662. private void createChildOrganism(Organism father, Organism mother) {
663.   // Create a new organism as a JSON object
664.   JSONObject newOrganism = new JSONObject(); // Creates a new JSON object for each
organism
665.
666.   // Adds a key/value pair to the json object initialised above
667.   newOrganism.put("organismId", organismCount);
668.   newOrganism.put("xCoordinate", mother.getxCoordinate()); // Spawns at its mother
669.   newOrganism.put("yCoordinate", mother.getyCoordinate());
670.   newOrganism.put("currentTerrain", GenerateMap.getTerrain(mother.getxCoordinate(),
mother.getyCoordinate()));
671.
672.   JSONObject statsObject = getStatsObject(father, mother); // Create the stats, will
need to take attributes from mother and father
673.
674.   newOrganism.put("stats", statsObject);
675.   newOrganism.put("sex", random.nextInt(0, 2)); // Random between male and female
676.   newOrganism.put("fatherID", father.getOrganismId());
677.   newOrganism.put("motherID", mother.getOrganismId());
678.   newOrganism.put("predator", false);
679.
680.   organismsToAdd.add(newOrganism); // Adding to the new organisms to be saved at the end
of the year
681.   organismCount++;
682.   if (father == mother) stats.setAgeLastBred(stats.getAge());
683. }
684.
685. // Sets the new stats to be representative of parental attributes
686. @SuppressWarnings("unchecked") // As it can be ignored
687. private JSONObject getStatsObject(Organism father, Organism mother) {
688.   JSONObject statsObject = new JSONObject(); // The stats object is initialised
689.
690.   double randomVariation = random.nextDouble(-0.2, 0.2); // Random variation to either
side
691.   int randomVariation2 = random.nextInt(-2, 2); // Random integer variation to either
side
692.   double chooseParent = random.nextInt(0, 2); // Choose to take from the mother or
father
693.
694.   // Random awareness from average of both parents with added double variation

```

```

695.         double newAwareness = (father.getStats().getCurrentAwareness() +
mother.getStats().getCurrentAwareness()) / 2.0;
696.         statsObject.put("currentAwareness", newAwareness + randomVariation);
697.
698.         statsObject.put("organismsEncountered", 0.0); // Updated during execution of program
699.         statsObject.put("awarenessHistory", new JSONArray()); // Only used during first 5
years, default value
700.         statsObject.put("age", 0); // The age needs to be representative and therefore start
at 0
701.
702.         statsObject.put("preferredFood", null); // Has to be set as FoodTypes however json
simple cannot set as type class. See below
703.
704.         FoodTypes newFood; // Chooses to eat food which either parent eats
705.         if (chooseParent == 1) newFood = father.getStats().getPreferredFood();
706.         else newFood = mother.getStats().getPreferredFood();
707.         statsObject.put("preferredFoodString", newFood.foodName); // Sets as a string to be
changed to a FoodType
708.
709.         statsObject.put("eatenFood", null); // Has not eaten yet, so does not need to be set
710.         statsObject.put("recentFood", new JSONArray()); // Used during first few years of
organisms life
711.         statsObject.put("foodFactor", 0.0); // Calculated after an organism eats
712.         statsObject.put("sizeFactor", 0.0); // Calculated after an organism eats
713.
714.         // An average size factor is calculated for both mother and father
715.         double averageFatherSizeFactor = father.getStats().getSizeFactor() / stats.getAge();
716.         double averageMotherSizeFactor = mother.getStats().getSizeFactor() / stats.getAge();
717.         double newSize = 0;
718.         if (averageFatherSizeFactor >= 0.25)
    newSize = newSize + 0.05; // These contribute to the increase or decrease in size
for the new organism
719.         else newSize = newSize - 0.05;
720.         if (averageMotherSizeFactor >= 0.25) newSize = newSize + 0.05;
721.         else newSize = newSize - 0.05;
722.         if (chooseParent == 1)
    newSize = newSize + father.getStats().getSize(); // Then chooses a random parent
to take size from
723.         else newSize = newSize + mother.getStats().getSize();
724.         statsObject.put("size", Double.parseDouble(decimalFormat.format(Math.max(0.3,
newSize)))); // Sets to 2dp
725.
726.
727.         // Takes the parents starting speed, and decreases if the current speed is low, and
visa-versa
728.         double newSpeed;
729.         if (chooseParent == 1) newSpeed = father.getStats().getStartSpeed();
730.         else newSpeed = mother.getStats().getStartSpeed();
731.         if (mother.getStats().getSpeed() > 10.0) newSpeed = newSpeed + 0.5;
732.         else newSpeed = newSpeed - 0.5;
733.         statsObject.put("speed", Math.min(30.0, Math.max(3.0, newSpeed))); // Lowest it can be
is 3, highest is 30.
734.
735.
736.         statsObject.put("startSpeed", newSpeed);
737.
738.         statsObject.put("currentTemp", 0); // Changes during the organisms life, no default
value
739.
740.         int newTemp; // Temp is taken from either father or mother, with an added integer
variation
741.         if (chooseParent == 1) newTemp = father.getStats().getPreferredTemp();
742.         else newTemp = mother.getStats().getPreferredTemp();
743.         statsObject.put("preferredTemp", newTemp + randomVariation2);
744.
745.         statsObject.put("recentTemp", new JSONArray()); // Changes during organisms life, no
default value
746.         statsObject.put("tempFactor", 1); // Changes during the organisms life, no default
value
747.
748.         String newTerrain; // Preferred terrain is taken from either father or mother
749.         if (chooseParent == 1) newTerrain = father.getStats().getPreferredTerrain();
750.         else newTerrain = mother.getStats().getPreferredTerrain();

```

```

751.         statsObject.put("preferredTerrain", newTerrain);
752.
753.         statsObject.put("secondPreferredTerrain", "Default"); // Will be chosen by the
organism itself
754.         statsObject.put("thirdPreferredTerrain", "Default"); // Will be chosen by the organism
itself
755.
756.         statsObject.put("health", 100.0); // Default value for all organisms
757.         statsObject.put("dead", false); // Default value for all organisms
758.         statsObject.put("ageLastBred", -1); // Default value for all organisms
759.
760.         // The generation of the organism set to the lowest of both parents + 1
761.         int generation = Math.min(father.getStats().getGeneration(),
mother.getStats().getGeneration());
762.         statsObject.put("generation", generation + 1);
763.
764.         statsObject.put("organismsMet", 0); // Default values
765.         statsObject.put("totalOrganismsMet", 0);
766.         statsObject.put("tsb", father.getStats().getTSB()); // Take the fathers TSB if it has
one
767.         statsObject.put("lastAttack", -1);
768.
769.     return statsObject; // Returns the stats as a JSONObject
770. }
771.
772. // This method ensures that the new organisms are added at the end of the year (when
organisms have finished updating)
773. // so that there are no discrepancies between GameWindow's organisms and the JSON file
774. private void addNewOrganisms(List<JSONObject> newOrganisms) {
775.     JSONArray jsonArray; // This will contain the current JSON array
776.     JSONParser parser = new JSONParser();
777.
778.     try (FileReader reader = new FileReader("organisms.json")) {
779.         // Parse the JSON file
780.         Object obj = parser.parse(reader);
781.         jsonArray = (JSONArray) obj;
782.
783.         jsonArray.addAll(newOrganisms); // Adding in the new organisms from the
organismsToAdd list
784.
785.         // Save the JSON array to the file
786.         FileWriter fileWriter = new FileWriter("organisms.json");
787.         fileWriter.write(jsonArray.toJSONString());
788.         fileWriter.flush();
789.     } catch (IOException | ParseException e) {
790.         throw new RuntimeException(e); // Catch any errors
791.     }
792. }
793.
794. // Checks if the organism should become asexual
795. // Checks the mothers of the organism for the number of fecund organisms encountered, if
it is less than 10 each, the organism can asexually reproduce
796. private void checkAsexual() {
797.     int generationsToCheck = 12; // The number of generations of mothers to check
798.     Organism currentOrganism = organisms[this.organismID]; // The organism being operated
on
799.
800.     // If the organism is water based then there is a significant boost in the chance it
becomes asexual
801.     if (Objects.equals(currentOrganism.getCurrentTerrain(), "Deep Water") ||
Objects.equals(currentOrganism.getCurrentTerrain(), "Water"))
802.         generationsToCheck = 3;
803.
804.     if (currentOrganism.getSex() != 2) { // If it is not already asexual
805.         // Check the generations of mothers for how many organisms have been encountered
806.         if (stats.getGeneration() >= generationsToCheck) {
807.             generationsToCheck = generationsToCheck - 3;
808.             int totalEncounters = 0;
809.             List<Organism> organismsToCheck = new ArrayList<>(); // List of mothers to
check
810.             for (int counter = 0; counter < generationsToCheck; counter++) {

```

```

811.             if (currentOrganism.getMotherID() != -1) {
812.                 organismsToCheck.add(organisms[currentOrganism.getMotherID()]); // Add
the immediate mother
813.                 currentOrganism = organisms[currentOrganism.getMotherID()]; // Set the
current organism to be the mother
814.             }
815.         }
816.         for (Organism organism : organismsToCheck) { // Add up the total encounters
817.             totalEncounters = totalEncounters + organism.getStats().getOrganismsMet();
818.         }
819.         if (totalEncounters <= 5 * generationsToCheck) { // If the number of
encounters are less than the 10 each
820.             organisms[this.organismID].setSex(2); // Set to be asexual
821.             GameWindow.consoleLog(this.organismID + " has become asexual");
822.             stats.setAgeLastBred(stats.getAge());
823.         }
824.     }
825. }
826. }
827.
828. // Checks if the organism should have terrain specific benefits
829. // Checks 5 generations of fathers, if they all have the same preferred terrain,
830. // then the organism will have terrain specific benefits (TSB) for that terrain
831. private void checkTSB() {
832.     int generationsToCheck = 8; // Number of generations to check
833.     Organism currentOrganism = organisms[this.organismID]; // The organism to be operated
on
834.     String preferredTerrain = currentOrganism.getStats().getPreferredTerrain(); // The
current organisms preferred terrain
835.     int terrainLog = 0; // Tracks how many of the fathers have the same preferred terrain
836.
837.     if (stats.getTSB() == null && stats.getGeneration() >= generationsToCheck) { // If the
TSB is not already set and organism is old enough
838.         generationsToCheck = generationsToCheck - 3;
839.         List<Organism> organismsToCheck = new ArrayList<>(); // The list of father
organisms to check
840.         for (int counter = 0; counter < generationsToCheck; counter++) { // Populates the
list
841.             if (currentOrganism.getFatherID() != -1) {
842.                 organismsToCheck.add(organisms[currentOrganism.getFatherID()]); // Adds
the father to the list
843.                 currentOrganism = organisms[currentOrganism.getFatherID()]; // Sets the
current one to operate on to the father of the previous
844.             }
845.         }
846.         for (Organism organism : organismsToCheck) { // Checks the preferred terrain of
each
847.             if (preferredTerrain.equals(organism.getStats().getPreferredTerrain())) {
848.                 terrainLog++; // Increments if the preferred terrain is the same
849.             }
850.         }
851.         if (terrainLog == organismsToCheck.size()) {
852.             stats.setTSB(preferredTerrain); // All preferred terrains are the same and
therefore organism will have TSB for that terrain
853.             GameWindow.consoleLog(currentOrganism.getOrganismID() + " has a new terrain
specific benefit on " + stats.getTSB());
854.         }
855.     }
856. }
857.
858. // This method checks if the organism should be a predator if it isn't already
859. private void checkPredation() {
860.     int generationsToCheck = 5; // Number of generations to check
861.     Organism currentOrganism = organisms[this.organismID]; // The organism to be operated
on
862.
863.     if (stats.getGeneration() >= generationsToCheck) { // If the organism is old enough
864.         int totalEncounters = 0; // To count the number of organisms met
865.         generationsToCheck = generationsToCheck - 3;
866.         List<Organism> organismsToCheck = new ArrayList<>(); // The list of father
organisms to check

```

```

867.         for (int counter = 0; counter < generationsToCheck; counter++) { // Populates the
list
868.             if (currentOrganism.getFatherID() != -1) {
869.                 organismsToCheck.add(organisms[currentOrganism.getFatherID()]); // Adds
the father to the list
870.                 currentOrganism = organisms[currentOrganism.getFatherID()]; // Sets the
current one to operate on to the father of the previous
871.             }
872.         }
873.         for (Organism organism : organismsToCheck) { // Add up the total encounters
874.             totalEncounters = totalEncounters +
organism.getStats().getTotalOrganismsMet();
875.         }
876.         if (totalEncounters >= 150 * generationsToCheck) { // If the number of encounters
are less than 450 each
877.             organisms[this.organismID].setPredator(true); // Organism becomes a predator
878.             GameWindow.consoleLog(this.organismID + " is now a predator");
879.         }
880.     }
881. }
882.
883. // The decision between which organism wins during an attack
884. private void newOrganismAttack(Organism attackingOrganism, Organism defendingOrganism) {
885.     double attackerSpeed, defenderSpeed, attackerSize, defenderSize; // Defining the
attributes to be used
886.     int attackerAge, defenderAge;
887.     int attackerHealth = 0, defenderHealth = 0; // Default health decrease
888.
889.     if (attackingOrganism.getStats().getLastAttack() <
attackingOrganism.getStats().getAge() - 15) {
890.         // Initialising the attributes
891.         attackerSpeed = attackingOrganism.getStats().getSpeed();
892.         attackerSize = attackingOrganism.getStats().getSize();
893.         attackerAge = attackingOrganism.getStats().getAge();
894.         defenderSpeed = defendingOrganism.getStats().getSpeed();
895.         defenderSize = defendingOrganism.getStats().getSize();
896.         defenderAge = defendingOrganism.getStats().getAge();
897.
898.         // Getting the differences between the attributes
899.         double speedScore = (attackerSpeed - defenderSpeed);
900.         double sizeScore = (attackerSize - defenderSize);
901.         int ageScore = (attackerAge - defenderAge);
902.
903.         if ((speedScore >= 0 && sizeScore >= 0) || (sizeScore >= 0 && ageScore < 20) ||
(speedScore >= 0 && ageScore < 20)) { // The attacker has won, give the attacker benefits
904.             attackerHealth = attackerHealth - random.nextInt(1, 4); // Attacker health
high decrease
905.             defenderHealth = defenderHealth - random.nextInt(7, 13); // Defender low
health decrease
906.             // Give the attacker an awareness increase
907.
organisms[attackingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[attackingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2);
908.             GameWindow.consoleLog(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and has won");
909.         } else { // The defendant has won, give the defendant benefits
910.             attackerHealth = attackerHealth - random.nextInt(4, 8); // Attacker health
high decrease
911.             defenderHealth = defenderHealth - random.nextInt(3, 7); // Defender low health
decrease
912.
organisms[defendingOrganism.getOrganismId()].getStats().setCurrentAwareness(organisms[defendingOrganism.getOrganismId()].getStats().getCurrentAwareness() + 0.2); // Give the defender an awareness
increase
913.             GameWindow.consoleLog(attackingOrganism.getOrganismId() + " has attacked " +
defendingOrganism.getOrganismId() + " and has lost");
914.         }
915.         // Increase their abilities and update the health
916.
organisms[attackingOrganism.getOrganismId()].getStats().setHealth(organisms[attackingOrganism.getOrganismId()].getStats().getHealth() + attackerHealth);

```

```

917. organisms[defendingOrganism.getOrganismId()].getStats().setHealth(organisms[defendingOrganism.getOrganismId()].getStats().getHealth() + defenderHealth);
918.
919. organisms[attackingOrganism.getOrganismId()].getStats().setLastAttack(organisms[attackingOrganism.getOrganismId()].getStats().getAge()); // Set the attacking organisms age to its current age
920. }
921. }

```

C1.11 SideMenu.java

```

1. import javax.swing.*;
2. import java.awt.event.MouseEvent;
3. import java.awt.event.MouseListener;
4. import java.net.URL;
5. import java.text.DecimalFormat;
6. import java.util.Objects;
7.
8. // The SideMenu class displays the organism information to the user and allows the user to
control
9. // the simulation
10. public class SideMenu implements MouseListener {
11.     private JPanel SideMenu; // The panel itself which contains all other components which are
listed below
12.     private JButton speedUpButton;
13.     private JButton slowDownButton;
14.     private JButton pauseGameButton;
15.     private JButton saveSimulationButton;
16.     private JButton loadPreviousSaveButton;
17.     private JButton exitToMainMenuButton;
18.     private JLabel organismCountLabel;
19.     private JLabel predatorCountLabel;
20.     private JLabel mostAware;
21.     private JLabel oldestOrganismLabel;
22.     private JLabel healthiestOrganismLabel;
23.     private JLabel fastestOrganismLabel;
24.     private JLabel averageTemperatureLabel;
25.     private JLabel mostPopulatedTerrainLabel;
26.     private JLabel leastPopulatedTerrainLabel;
27.     private JLabel newOrganismsSinceYearLabel;
28.     private JLabel currentYearLabel;
29.     private JLabel currentGameSpeedLabel;
30.     private JPanel organismInformationPanel;
31.     private JTextField organismToFind;
32.     private JButton findButton;
33.     private JTabbedPane organismInfoTab;
34.     private JCheckBox showDeadOrganismsCheckBox;
35.     private JLabel KeyLabel;
36.     private JCheckBox showConsoleCheckbox;
37.     private static final DecimalFormat decimalFormat = new DecimalFormat("#.##");
38.
39.     // Returns the side menu panel to be added to the side of the game window
40.     public JPanel getSideMenu() {

```

```

41.         return SideMenu; // Returns the panel (with added components)
42.     }
43.
44.     // The constructor for SideMenu
45.     // Initialises all buttons, text, and adds in the key once it is called at the start of
the simulation
46.     public SideMenu(){
47.         // Setting text such as the current game speed and year from GameWindow
48.         currentGameSpeedLabel.setText("Current Game Speed: " + (double) GameWindow.timeFactor
/ 1000 + " seconds/year");
49.         currentYearLabel.setText("Current Year: " + GameWindow.years);
50.
51.         slowDownButton.addMouseListener(this); // Adding relevant mouse listeners
52.         speedUpButton.addMouseListener(this);
53.         pauseGameButton.addMouseListener(this);
54.         exitToMainMenuButton.addMouseListener(this);
55.         saveSimulationButton.addMouseListener(this);
56.         findButton.addMouseListener(this);
57.         loadPreviousSaveButton.addMouseListener(this);
58.
59.         ImageIcon keyImage; // The key image to be added to the side menu
60.         URL resourceUrl = SideMenu.class.getResource("/KeyResource.png"); // Attempts to find
from class (jar)
61.         if (resourceUrl != null) keyImage = new ImageIcon(resourceUrl);
62.         else keyImage = new
ImageIcon("C:/Users/Harry/IdeaProjects/EvSim/res/KeyResource.png"); // Else get from source file
(IDE)
63.
64.         KeyLabel.setIcon(keyImage); // Adds in the image as an icon
65.     }
66.
67.     // This function updates the UI to show either global or local organism statistics.
68.     // Takes the organisms' statistics from JSON, and updates for the relevant label
69.     // Refer to comments for explanation of specific sections
70.     public void UpdateUI() { // Called each time the game window paints to the screen
71.         GenerateOrganisms.showDeadOrganisms = showDeadOrganismsCheckBox.isSelected(); // /
Update whether to show the dead organisms
72.
73.         if(GameWindow.gamePaused) pauseGameButton.setText("Play Game"); // Update the play / pause
button (to show what it is currently)
74.         else pauseGameButton.setText("Pause Game");
75.
76.         // If the user wishes to show the console, then show it, if not, remove it
77.         if(showConsoleCheckbox.isSelected()){
78.             GameWindow.showConsole = true;
79.         } else {
80.             GameWindow.showConsole = false;
81.             GameWindow.frame.remove(GameWindow.consolePanel);
82.         }
83.
84.         // Initialising attributes that are to be used
85.         int oldestOrganism = 0, healthiestOrganism = 0, fastestOrganism = 0, averageTemp,
mostAwareOrganism = 0, predatorCount = 0;
86.         String mostCommonTerrain = "N/A", leastCommonTerrain = "N/A";
87.
88.         // If an organism is not selected, display global statistics
89.         if (GameWindow.organismSelected == -1) {
90.             // The length is equivalent to the number of objects and therefore the number of
organisms
91.             // Once organisms have the ability to die, this will have to update to only count
alive organisms
92.             organismCountLabel.setText("Total Organism Count: " +
GameWindow.organisms.length);
93.
94.             // Temporary variables which are used to compare organisms
95.             int tempAge = GameWindow.organisms[0].getStats().getAge();
96.             double tempHealth = GameWindow.organisms[0].getStats().getHealth();
97.             double tempSpeed = GameWindow.organisms[0].getStats().getSpeed();
98.             double tempAwareness = GameWindow.organisms[0].getStats().getCurrentAwareness();
99.             double totalTemp = 0;

```

```

100.         int grassCount = 0, waterCount = 0, mountainCount = 0, beachCount = 0, snowCount =
101. 0, forestCount = 0, deepWaterCount = 0;
102.         // Loops through each organism and compares statistics
103.         // Refer to comments for explanation of specific sections
104.         for (Organism organism : GameWindow.organisms) {
105.             // Finds the organism with the highest age
106.             if (tempAge < organism.getStats().getAge()) {
107.                 oldestOrganism = organism.getOrganismId();
108.                 tempAge = organism.getStats().getAge();
109.             }
110.             // Finds the organism with the highest health
111.             if (tempHealth < organism.getStats().getHealth()) {
112.                 healthiestOrganism = organism.getOrganismId();
113.                 tempHealth = organism.getStats().getHealth();
114.             }
115.             // Finds the organism with the highest speed
116.             if (tempSpeed < organism.getStats().getSpeed()) {
117.                 fastestOrganism = organism.getOrganismId();
118.                 tempSpeed = organism.getStats().getSpeed();
119.             }
120.             if (tempAwareness < organism.getStats().getCurrentAwareness()) {
121.                 mostAwareOrganism = organism.getOrganismId();
122.                 tempAwareness = organism.getStats().getCurrentAwareness();
123.             }
124.
125.             // Adds the preferred temperature of each organism
126.             // Is used to get the mean temperature
127.             totalTemp = totalTemp + GameWindow.organisms[0].getStats().getCurrentTemp();
128.
129.             // To find how many organisms are in each terrain
130.             switch (organism.getCurrentTerrain()) {
131.                 case "Grass" -> grassCount++;
132.                 case "Water" -> waterCount++;
133.                 case "Mountain" -> mountainCount++;
134.                 case "Beach" -> beachCount++;
135.                 case "Snow" -> snowCount++;
136.                 case "Forest" -> forestCount++;
137.                 case "Deep Water" -> deepWaterCount++;
138.             }
139.
140.             if (organism.isPredator()) predatorCount++; // Total the number of predators
141.         }
142.
143.         // Variables which are used to compare how common terrains are, as well as set the
144.         // equivalent terrain name
145.         int[] terrainCounts = {grassCount, waterCount, mountainCount, beachCount,
146. snowCount, forestCount, deepWaterCount};
147.         String[] terrainNames = {"Grass", "Water", "Mountain", "Beach", "Snow", "Forest",
148. "Deep Water"};
149.         int maxCount = 0, minCount = Integer.MAX_VALUE; // Set to MAX_VALUE so that there
150.         can be a large amount of organisms in a terrain, will never reach this value
151.         // Checks for both the most common and least common terrain
152.         for (int i = 0; i < terrainCounts.length; i++) {
153.             if (terrainCounts[i] > maxCount) { // If there are more organisms than the
154. previous
155.                 maxCount = terrainCounts[i]; // Current highest organism count
156.                 mostCommonTerrain = terrainNames[i]; // Sets the terrain name as the most
157. common
158.             }
159.             if (terrainCounts[i] < minCount) { // If there are fewer organisms than the
160. previous
161.                 minCount = terrainCounts[i]; // Current lowest organism count
162.                 leastCommonTerrain = terrainNames[i]; // Sets the terrain name as the
163. least common
164.             }
165.         }
166.         averageTemp = (int) (totalTemp / GameWindow.organisms.length); // Calculates the
167. mean temperature (average)

```

```

161.          // Updates each label to show the relevant information, as well as the statistic
162.          to go with it
163.          oldestOrganismLabel.setText("<html>Oldest Organism ID: " + oldestOrganism + "
<u>(" + GameWindow.organisms[oldestOrganism].getStats().getAge() + " years)</u></html>");
164.          predatorCountLabel.setText("Predator Count: " + predatorCount);
165.          mostAware.setText("<html>Most Aware: " + mostAwareOrganism + " (<u>" +
Double.parseDouble(decimalFormat.format(tempAwareness * 10)) + "px</u>)</html>");
166.          healthiestOrganismLabel.setText("<html>Healthiest Organism ID: " +
healthiestOrganism + " <u>(" + GameWindow.organisms[healthiestOrganism].getStats().getHealth() +
")</u></html>\"");
167.          fastestOrganismLabel.setText("<html>Fastest Organism ID: " + fastestOrganism + "
<u>(" + GameWindow.organisms[fastestOrganism].getStats().getSpeed() + "px/year)</u></html>");
168.          averageTemperatureLabel.setText("<html>Average Temperature: " + " <u>(" +
averageTemp + ")</u></html>\"");
169.          mostPopulatedTerrainLabel.setText("<html>Most Popular: <i>" + mostCommonTerrain +
"</i> <u>(" + maxCount + " organisms)</u></html>");
170.          leastPopulatedTerrainLabel.setText("<html>Least Popular: <i>" + leastCommonTerrain +
"</i> <u>(" + minCount + " organisms)</u></html>");
171.          newOrganismsSinceYearLabel.setText("<html>New Organisms: " + " <u>(" +
(GameWindow.organisms.length - GenerateOrganisms.organismCount) + ")</u></html>\"");
172.
173.      } else { // There is an organism selected, show its attributes
174.
175.          // If there is no preferred food, show it as N/A, otherwise show the name of the
preferred food
176.          String preferredFood;
177.          if(GameWindow.organisms[GameWindow.organismSelected].getStats().getPreferredFood() ==
null) preferredFood = "N/A";
178.          else preferredFood =
GameWindow.organisms[GameWindow.organismSelected].getStats().getPreferredFood().foodName;
179.
180.          // Returning the sex of the organism as a String
181.          String sex = switch (GameWindow.organisms[GameWindow.organismSelected].getSex()){
182.              case 0 -> "Female";
183.              case 1 -> "Male";
184.              case 2 -> "Asexual";
185.              default -> "N/A"; // Has to have a default case (if no sex then just show N/A)
186.          };
187.
188.          // Replace the labels with specific organism statistics
189.          organismCountLabel.setText("Total Organism Count: " +
GameWindow.organisms.length); // The number of organisms
190.          predatorCountLabel.setText("Selected Organism ID: " +
GameWindow.organisms[GameWindow.organismSelected].getOrganismId()); // The organism selected
currently
191.          // The coordinates of the organism (x, y) currentTerrain
192.          mostAware.setText("<html>Coordinates: (" +
GameWindow.organisms[GameWindow.organismSelected].getxCoordinate() + ", " +
GameWindow.organisms[GameWindow.organismSelected].getyCoordinate() + ")"
193.          <i>" +
194.          GameWindow.organisms[GameWindow.organismSelected].getCurrentTerrain() +
"</i></html>\"");
195.          // Whether it is a predator or not and if it has a terrain specific benefit
196.          oldestOrganismLabel.setText("<html>Predator: " +
GameWindow.organisms[GameWindow.organismSelected].isPredator() +
"<br>Terrain Specific Benefit: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getTSB() + "</html>");
198.          // The preferred terrain of the organism if available
199.          healthiestOrganismLabel.setText("Preferred Terrain: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getPreferredTerrain());
200.          fastestOrganismLabel.setText("Sex: " + sex); // The sex of the organism (see above
for sex initialisation)
201.
202.          // Displays the other stats in HTML bullet point format
203.          averageTemperatureLabel.setText("<html><p>Stats: </p>" +
204.              "<ul>" +
205.                  "<li>Age: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getAge() + "</li>" + // Age
206.                  "<li>Size: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getSize() + "sq/m</li>" + // Size

```

```

207.           "      <li>Awareness: " +
decimalFormat.format(GameWindow.organisms[GameWindow.organismSelected].getStats().getCurrentAwareness() * 10) + "px</li>" + // Awareness
208.           "      <li>Speed: " + (int)
GameWindow.organisms[GameWindow.organismSelected].getStats().getSpeed() + "px / year</li>" + // Speed
209.           "      <li>Health: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getHealth() + "</li>" + // Health
210.           "      <li>Current Temperature: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getCurrentTemp() + "</li>" + // Current Temperature
211.           "      <li>Preferred Temperature: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getPreferredTemp() + "</li>" + // Preferred Temperature
212.           "</ul></html>");
213.
214.           // The parents of the organism (father, mother)
215.           mostPopulatedTerrainLabel.setText("Parent IDs: (" +
GameWindow.organisms[GameWindow.organismSelected].getFatherID() + "f, " +
GameWindow.organisms[GameWindow.organismSelected].getMotherID() + "m)");
216.
217.           // If the organism has eaten, display the eaten food
218.
if(GameWindow.organisms[GameWindow.organismSelected].getStats().getEatenFood().foodName != null)
leastPopulatedTerrainLabel.setText("Eaten Food: " +
GameWindow.organisms[GameWindow.organismSelected].getStats().getEatenFood().foodName);
219.           else leastPopulatedTerrainLabel.setText("Eaten Food: N/A"); // Otherwise display
N/A
220.           // Displays the organisms preferred food
221.           newOrganismsSinceYearLabel.setText("Preferred Food: " + preferredFood);
222.       }
223.
224.           // Displays the years held in GameWindow and the season held in Organisms at the top
of the menu
225.           currentYearLabel.setText("Current Year: " + GameWindow.years + " - Season: " +
Organisms.season);
226.       }
227.
228.           // This checks for any mouse clicks by the user
229.           // On components which have a mouse listener, such as the save and load buttons
230.           @Override
231.           public void mouseClicked(MouseEvent mouseEvent) {
232.               // Every 1000 is a second, anything longer than 8.5 seconds will be too long.
233.               // Anything below 0.5 seconds is too short.
234.               if(mouseEvent.getSource() == slowDownButton && GameWindow.timeFactor < 8500)
GameWindow.timeFactor = GameWindow.timeFactor + 500; // Increase the seconds per year by 0.5
seconds
235.               if(mouseEvent.getSource() == speedUpButton && GameWindow.timeFactor > 500)
GameWindow.timeFactor = GameWindow.timeFactor - 500; // Decrease the seconds per year by 0.5
seconds
236.               if(mouseEvent.getSource() == pauseGameButton) {
237.                   GameWindow.gamePaused = !GameWindow.gamePaused; // Flip the paused boolean
238.                   if(Objects.equals(pauseGameButton.getText(), "Play Game"))
pauseGameButton.setText("Pause Game");
239.                   else pauseGameButton.setText("Play Game"); // Flip the text shown to equate
whether the game is paused or not
240.               }
241.
242.               // Disallow the user from setting anything lower than 0.5 seconds/year
speedUpButton.setEnabled(GameWindow.timeFactor != 500);
244.               // Disallow the user from setting anything higher than 8.5 seconds/year
slowDownButton.setEnabled(GameWindow.timeFactor != 8500);
246.
247.               // For every 1000 in gameWindow.timeFactor, it is 1 second. Therefore, to divide it by
1000 is the time in seconds.
248.               currentGameSpeedLabel.setText("Current Game Speed: " + (double) GameWindow.timeFactor
/ 1000 + " seconds/year"); // Update the label to show the current game speed in seconds per year
249.
250.               // If the user wishes to exit to the main menu
251.               if(mouseEvent.getSource() == exitToMainMenuButton){
252.                   // Reset all attributes

```

```

253.         GameWindow.organismSelected = -1;
254.         new OpeningMenu(); // Return to the opening menu
255.         GameWindow.years = 0;
256.         GameWindow.timeFactor = 2500;
257.         GameWindow.gamePaused = false;
258.         GameWindow.thread.interrupt(); // Stop from updating and running
259.         GameWindow.frame.dispose(); // Remove the game window
260.     }
261.
262.     // Load and save simulation buttons
263.     if(mouseEvent.getSource() == saveSimulationButton){
264.         new SaveSimulation(); // Load the save simulation window
265.         GameWindow.gamePaused = true; // Pause the game (usability)
266.     }
267.     // Load simulation button
268.     if(mouseEvent.getSource() == loadPreviousSaveButton){
269.         new LoadSimulation(); // Load the load simulation window
270.         GameWindow.gamePaused = true; // Pause the game (usability)
271.     }
272.
273.     // If the found button was pressed (find organism)
274.     if(mouseEvent.getSource() == findButton){
275.         try { // Attempt to pass the input into an integer
276.             GameWindow.organismSelected =
277.                 Integer.parseInt(organismToFind.getText().trim()); // Validate and set the input
278.             organismInfoTab.setSelectedIndex(0); // Return to the organism information tab
279.             organismToFind.setText(""); // Clear the text from the text box
280.         } catch (NumberFormatException e) { // If it could not be validated create the
281.             error message
282.             new ErrorWindow("Organism not found!");
283.         }
284.     }
285.     @Override
286.     public void mousePressed(MouseEvent e) {}
287.     @Override
288.     public void mouseReleased(MouseEvent e) {}
289.     @Override
290.     public void mouseEntered(MouseEvent e) {}
291.     @Override
292.     public void mouseExited(MouseEvent e) {}
293. }

```

C1.12 SaveSimulation.java

```

1. import org.json.simple.JSONObject;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.MouseEvent;
5. import java.awt.event.MouseListener;
6. import java.io.FileWriter;
7. import java.io.IOException;
8. import java.nio.file.Files;
9. import java.nio.file.Paths;
10. import java.time.LocalDateTime;
11. import java.time.format.DateTimeFormatter;
12. import java.nio.file.Path;
13.
14. // This class allows the user to save the simulation at its current state.
15. // When it is called, it allows the user to select a name to save as,

```

```

16. // before saving it physically
17. public class SaveSimulation implements MouseListener {
18.     private final JFrame frame; // The window containing the save options
19.     private JPanel panel1; // The panel containing other components listed below
20.     private JTextField saveNameTextField;
21.     private JButton cancelButton;
22.     private JButton saveSimulationButton;
23.
24.     // The constructor for SaveSimulation
25.     // Opens the window, and places placeholder text holding the default name for a new file
26.     // (current date and time)
26.     public SaveSimulation(){
27.         LocalDateTime currentDateTime = LocalDateTime.now(); // To find the current date and
28.         // time (default file name)
29.         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd_HHmm"); // // Formatting the current date and time)
30.         String defaultSaveName = currentDateTime.format(formatter); // For example
30.         20231216_1311
31.         saveNameTextField.setText(defaultSaveName); // Sets it to be the placeholder
32.
33.         frame = new JFrame("Save Simulation"); // The window name
34.         frame.setPreferredSize(new Dimension(300,175)); // The window size
35.         frame.add(panel1); // As the panel already contains the text, labels and buttons, only
36.         // the panel has to be added to the frame.
37.         frame.setResizable(false);
38.         frame.pack();
39.         frame.setLocationRelativeTo(null);
40.         frame.setVisible(true);
41.
42.         cancelButton.addMouseListener(this); // Adding the required mouse listeners to listen
43.         for a button press
44.         saveSimulationButton.addMouseListener(this);
45.     }
46.
47.     // This class listens for components that have been clicked (which have a listener)
48.     @Override
49.     @SuppressWarnings("unchecked") // As this warning can be ignored
50.     public void mouseClicked(MouseEvent e) {
51.         // If the user chooses to save the simulation
52.         if(e.getSource() == saveSimulationButton) {
53.             String saveFileName = saveNameTextField.getText(); // The name of the folder and
54.             // JSON file to save as
55.             // File to be copied
56.             Path sourcePath = Paths.get("./organisms.json");
57.
58.             // Directory path for Save Games
59.             Path saveGamesDirectory = Paths.get("./Save Games");
60.
61.             // The path of the directory to create - ./Save Games/saveFileName
62.             Path destinationDirectory = saveGamesDirectory.resolve(saveFileName);
63.
64.             // The path of the file to create - ./Save Games/saveFileName/organisms.json
65.             Path finalDestination = destinationDirectory.resolve("organisms.json");
66.
67.             // The second JSON file content containing the year, seed etc
68.             JSONObject jsonObject = new JSONObject(); // The JSON which will be saved
69.             jsonObject.put("seed", PerlinNoise.getSeed()); // The current seed
70.             jsonObject.put("currentYear", GameWindow.years); // The current year
71.             jsonObject.put("windowHeight", GameWindow.HEIGHT); // The window height and width
72.             jsonObject.put("windowWidth", GameWindow.WIDTH);
73.             jsonObject.put("predatorPreyEnabled", Organisms.predatorPreyEnabled); // Whether
74.             // predator prey is enabled
75.             jsonObject.put("seasonsEnabled", Organisms.seasonsEnabled);
76.             jsonObject.put("currentSeason", Organisms.season);
77.             try {
78.                 if(!Files.exists(destinationDirectory) || !Files.exists(finalDestination)) {
// Ensures no error when 2 saves have the same name
79.                     // Create the directory
80.                     Files.createDirectories(destinationDirectory);

```

```

78.                                // Copies the ./organisms.json into ./Save
Games/saveFileName/organisms.json
79.                                Files.copy(sourcePath, finalDestination);
80.
81.                                // Write the second JSON file
82.                                FileWriter fileWriter = new FileWriter("./Save Games/" + saveFileName +
"/" + saveFileName + ".json");
83.                                fileWriter.write(jsonObject.toJSONString());
84.                                fileWriter.flush();
85.
86.                                frame.dispose(); // Removing the frame
87.                                GameWindow.gamePaused = false;
88.                                ErrorWindow.windowName = "Success";
89.                                new ErrorWindow("<html><p style=\"text-align:center;\">Simulation saved
as:<br>" + saveFileName + "</p></html>"); // Displaying a success message
90.                            } else {
91.                                new ErrorWindow("That file name is already in use"); // If the file
already exists, display to the user
92.                            }
93.                        } catch (IOException error) {
94.                            throw new RuntimeException(error); // Log any errors
95.                        }
96.                    }
97.
98.                    if(e.getSource() == cancelButton){ // Else if they choose to cancel resume the game
and close the window
99.                        GameWindow.gamePaused = false; // Resume the game
100.                       frame.dispose(); // Dispose of the current window
101.                   }
102.               }
103.
104.               // Not used, but required by the program
105.               @Override
106.               public void mousePressed(MouseEvent e) {}
107.
108.               @Override
109.               public void mouseReleased(MouseEvent e) {}
110.
111.               @Override
112.               public void mouseEntered(MouseEvent e) {}
113.
114.               @Override
115.               public void mouseExited(MouseEvent e) {}
116.           }

```

C1.13 LoadSimulation.java

```

1. import com.fasterxml.jackson.databind.ObjectMapper;
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.MouseEvent;
5. import java.awt.event.MouseListener;
6. import java.io.File;

```

```

7. import java.io.IOException;
8. import java.nio.file.*;
9. import java.time.ZoneId;
10. import java.time.Instant;
11. import java.time.LocalDateTime;
12. import java.time.format.DateTimeFormatter;
13. import java.util.regex.Matcher;
14. import java.util.regex.Pattern;
15.
16. // This allows an old save to be loaded with the appropriate settings being carried over
17. // Code for linking the load menu as well as the load function itself (see mouseClicked)
18. public class LoadSimulation implements MouseListener {
19.     private JButton loadSaveButton;
20.     private JButton cancelButton;
21.     private JPanel panel1; // The panel containing the form itself
22.     private JList<String> list1;
23.     private JButton deleteButton;
24.     private JButton renameButton;
25.     public static JFrame frame; // The window for the load menu
26.
27.     // The constructor of LoadSimulation which displays the window itself
28.     // which contains the list of save files
29.     public LoadSimulation() {
30.         // This method returns a list model which holds all the names of the saves, as a
component which can be displayed on the JList
31.         String path = "./Save Games"; // Path to the parent folder
32.
33.         DefaultListModel<String> listModel = new DefaultListModel<>(); // List model storing
the save names and details
34.
35.         File folder = new File(path); // Parent folder
36.         File[] folders = folder.listFiles(File::isDirectory); // Array of all child folders
37.
38.         // Loops through folders and adds to list
39.         if (folders != null) {
40.             for (File subFolder : folders) {
41.                 // Create a date and time for the last modified date
42.                 LocalDateTime lastModifiedDateTime = LocalDateTime.ofInstant(
43.                     Instant.ofEpochMilli(subFolder.lastModified()), // Time of last
modification
44.                     ZoneId.systemDefault() // Time zone
45.                 );
46.                 // Bullet points and formats date last modified
47.                 listModel.addElement("<html><ul><li>" + subFolder.getName() + " Last Used: " +
lastModifiedDateTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy")) + "</li></ul></html>");
48.             }
49.         }
50.
51.         list1.setModel(listModel); // Add the filled model to the JList
52.
53.         renameButton.setEnabled(false); // Setting to be disabled by default, and adding mouse
listeners
54.         deleteButton.setEnabled(false);
55.         loadSaveButton.setEnabled(false);
56.         renameButton.addMouseListener(this); // MouseListeners to check if the user clicks on
the button
57.         deleteButton.addMouseListener(this);
58.         loadSaveButton.addMouseListener(this);
59.         cancelButton.addMouseListener(this);
60.         list1.addMouseListener(this);
61.
62.         frame = new JFrame("Load Simulation"); // The frame itself
63.         frame.setPreferredSize(new Dimension(500, 540)); // Ideal size
64.         frame.add(panel1); // panel1 contains all other components
65.         frame.setResizable(false); // So no size issues
66.         frame.pack(); // So all is correct size
67.         frame.setLocationRelativeTo(null); // Central
68.         frame.setVisible(true);
69.     }
70.
71.     // Checks for the user clicking on components which have a mouseListener

```

```

72.    // In this case the JList and the JButtons
73.    @Override
74.    public void mouseClicked(MouseEvent e) {
75.
76.        if(e.getSource() == list1){ // If the user clicks on an object in the list, enable the
buttons
77.            renameButton.setEnabled(true); // Allow the other buttons to become functional
78.            deleteButton.setEnabled(true);
79.            loadSaveButton.setEnabled(true);
80.        }
81.
82.        // Load the simulation under the selected file name
83.        if(e.getSource() == loadSaveButton && loadSaveButton.isEnabled()) {
84.            String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
85.            ObjectMapper objectMapper = new ObjectMapper();
86.            try {
87.                // Replace organisms.json with the stored json file
88.                Files.copy(Paths.get("./Save Games/" + currentlySelectedFile +
"/organisms.json"), Paths.get("./organisms.json"), StandardCopyOption.REPLACE_EXISTING);
89.                // Load the stored settings
90.                Settings newLoad = objectMapper.readValue(new File("./Save Games/" +
currentlySelectedFile + "/" + currentlySelectedFile + ".json"), Settings.class); // Creating an
array of organisms from organisms.json
91.                // start a new game. Sends loaded save as true to indicate what to load. Sends
the newLoad to be loaded
92.                if(GameWindow.frame != null) {
93.                    GameWindow.frame.dispose();
94.                    GameWindow.thread.interrupt();
95.                }
96.                GameWindow newGameWindow = new GameWindow(newLoad.getWindowWidth(),
newLoad.getWindowHeight(), true, newLoad);
97.                newGameWindow.startGameThread(); // Start the thread
98.                frame.dispose(); // Dispose of the load menu
99.                OpeningMenu.frame.dispose(); // Dispose of the opening menu
100.            } catch (IOException ex) {
101.                throw new RuntimeException(ex);
102.            }
103.        }
104.
105.        // Deletes a save (folder and JSON files)
106.        if(e.getSource() == deleteButton && deleteButton.isEnabled()) {
107.            String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
108.
109.            // Removes selected save from the list on the screen
110.            DefaultListModel<String> listModel = (DefaultListModel<String>) list1.getModel();
111.            listModel.remove(list1.getSelectedIndex());
112.
113.            // Deletes the folder itself
114.            try {
115.                // Deletes the folder and its contents recursively
116.                deleteFolder(Paths.get("./Save Games/" + currentlySelectedFile));
117.                frame.dispose(); // Refresh the load save page
118.                new LoadSimulation();
119.            } catch (IOException error) {
120.                throw new RuntimeException(error);
121.            }
122.        }
123.
124.        // Renames a save file
125.        if(e.getSource() == renameButton && renameButton.isEnabled()) {
126.            String currentlySelectedFile = currentSelectedFile(); // The name of the currently
selected file
127.
128.            // Gets the name that the user wishes to rename to, and renames in the class
renameFile
129.            new RenameFile(currentlySelectedFile);
130.        }
131.
132.        // Close the frame

```

```

133.     if(e.getSource() == cancelButton) {
134.         frame.dispose();
135.         GameWindow.gamePaused = false;
136.     }
137. }
138.
139. // The folder cannot be deleted if it has contents in it, therefore this method is needed
140. // to
141. private static void deleteFolder(Path folderPath) throws IOException {
142.     // 'Walk' the file tree to delete each file starting from the given folder path
143.     Files.walk(folderPath).sorted((p1, p2) -> -p1.compareTo(p2)).forEach(path -> {
144.         try {
145.             Files.delete(path); // Deletes the sub files/folders, in this case
146.             only the sub-files
147.         } catch (IOException e) {
148.             throw new RuntimeException(e); // Log any errors
149.         }
150.     });
151.
152. // Finds the name of the currently selected file from the JList
153. private String currentSelectedFile(){
154.     String currentlySelectedFile = "";
155.     String htmlString = list1.getSelectedValue(); // The selected value (which returns an
HTML string)
156.
157.     // Uses a pattern and a matcher to extract the name of the file from the HTML string
158.     Pattern pattern = Pattern.compile("<li>(.*)? Last Used:");
159.     Matcher matcher = pattern.matcher(htmlString);
160.
161.     if (matcher.find()) currentlySelectedFile = matcher.group(1); // If the pattern is
found then the file name can be updated
162.
163.     return currentlySelectedFile; // Return the name of the file
164. }
165.
166. // Not used but still needed within the program
167. @Override
168. public void mousePressed(MouseEvent e) {}
169.
170. @Override
171. public void mouseReleased(MouseEvent e) {}
172.
173. @Override
174. public void mouseEntered(MouseEvent e) {}
175.
176. @Override
177. public void mouseExited(MouseEvent e) {}
178. }

```

C1.14 RenameFile.java

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.MouseEvent;
4. import java.awt.event.MouseListener;
5. import java.io.IOException;
6. import java.nio.file.Files;
7. import java.nio.file.Paths;
8.
9. // This allows a save file to be renamed to the users choice of new name
10. // Once the button is pressed on the LoadSimulation menu, this frame
11. // opens and the user is asked what name they would like to change to.
12. // This class handles that operation.
13. public class RenameFile implements MouseListener {
14.     private JPanel panel1; // The panel containing the components below
15.     private JTextField renameNameTextField;
16.     private JButton cancelButton;

```

```

17.     private JButton renameFileButton;
18.     private final JFrame frame; // The window which contains the menu
19.     private final String currentlySelectedFile; // The file which is selected by the user
20.
21.     // The constructor for RenameFile, fetches the name of the selected file, and opens the
window
22.     // which is set as a placeholder for the text field renameNameTextField
23.     public RenameFile(String originalFileName) {
24.         currentlySelectedFile = originalFileName; // Stores the old string to be used to find
the file
25.         renameNameTextField.setText(originalFileName); // Sets the placeholder as the current
file name
26.
27.         frame = new JFrame("Rename File"); // Window title
28.         frame.setPreferredSize(new Dimension(300,175)); // Size of the window
29.         frame.add(panel1); // As the panel already contains the text, labels and buttons, only
the panel has to be added to the frame.
30.         frame.setResizable(false);
31.         frame.pack();
32.         frame.setLocationRelativeTo(null);
33.         frame.setVisible(true);
34.
35.         cancelButton.addMouseListener(this); // Add required mouse listeners
36.         renameFileButton.addMouseListener(this);
37.     }
38.
39.     // This detects the user clicking the mouse on components with a mouse listnener, in this
case
40.     // it is listening for the buttons being clicked.
41.     @Override
42.     public void mouseClicked(MouseEvent e) {
43.         // If they choose to rename the file, find the old files and rename appropriately
44.         if(e.getSource() == renameFileButton){
45.             String newFileName = renameNameTextField.getText(); // The new name
46.
47.             if(newFileName.isEmpty()){ // Ensure the user has not entered an empty string
48.                 new ErrorWindow("Please enter a name! It cannot be blank"); // Display an
appropriate message if so
49.             } else {
50.                 try { // Renames the file and folder
51.                     // First path is the json file containing seed and years, second is the
same file but what to name it as
52.                     Files.move(Paths.get("./Save Games/" + currentlySelectedFile + "/" +
currentlySelectedFile + ".json"), Paths.get("./Save Games/" + currentlySelectedFile + "/" +
newFileName + ".json"));
53.                     // First path is the folder, second is the new name for the folder
54.                     Files.move(Paths.get("./Save Games/" + currentlySelectedFile),
Paths.get("./Save Games/" + newFileName));
55.
56.                     LoadSimulation.frame.dispose(); // remove the old load page
57.                     frame.dispose(); // Remove the rename page
58.                     new LoadSimulation(); // Bring back the load page with the updated names
59.                 } catch (IOException ex) {
60.                     throw new RuntimeException(ex); // Print any error
61.                 }
62.             }
63.         }
64.         // If they choose to cancel, just remove the window and do no other updates
65.         if(e.getSource() == cancelButton){
66.             frame.dispose(); // Remove the window
67.         }
68.     }
69.
70.     // Not used, but needed by the program
71.     @Override
72.     public void mousePressed(MouseEvent e) {}
73.
74.     @Override
75.     public void mouseReleased(MouseEvent e) {}
76.
77.     @Override

```

```

78.     public void mouseEntered(MouseEvent e) {}
79.
80.     @Override
81.     public void mouseExited(MouseEvent e) {}
82. }
```

C1.15 ErrorWindow.java

```

1. import javax.swing.*;
2. import java.awt.*;
3.
4. // This class is used to display information on demand, such as an error or a success message.
5. // Is used to display any message, very useful for giving the user information
6. public class ErrorWindow {
7.     public static String windowName = "Error"; // Displays as an error by default, but can be
changed to success or anything else
8.     public ErrorWindow(String errorMessage){ // Takes the desired error message as a string
9.         JFrame frame = new JFrame(windowName);
10.        frame.setPreferredSize(new Dimension(250,150));
11.        JLabel errorLabel = new JLabel(); // Creates a new JLabel which displays the string
12.        errorLabel.setText(errorMessage); // Adds the error message to the created JLabel
13.        errorLabel.setHorizontalAlignment(SwingConstants.CENTER); // Center-align the text
14.        frame.add(errorLabel); // Adds this to the centre of the frame
15.        frame.pack();
16.        frame.setResizable(false);
17.        frame.setLocationRelativeTo(null);
18.        frame.setVisible(true);
19.    }
20. }
```

C2 Models

C2.1 Colours.java

```

1. public class Colours {
2.     private int red; // Initialising all variables with the same name and type as stored in
JSON
3.     private int green;
4.     private int blue;
5.     private String setting;
6.     // Getter and setter methods
```

```

7.     public int getRed() {
8.         return red;
9.     }
10.    public void setRed(int red) {
11.        this.red = red;
12.    }
13.    }
14.    public int getGreen() {
15.        return green;
16.    }
17.    }
18.    public void setGreen(int green) {
19.        this.green = green;
20.    }
21.    }
22.    public int getBlue() {
23.        return blue;
24.    }
25.    }
26.    public void setBlue(int blue) {
27.        this.blue = blue;
28.    }
29.    }
30.    public String getSetting() {
31.        return setting;
32.    }
33.    }
34.    public void setSetting(String setting) {
35.        this.setting = setting;
36.    }
37.    }
38. }

```

C2.2 Options.java

```

1. import java.util.List;
2. public class Options {
3.     private int WindowHeight; // Initialising all variables with the same name and type as
stored in JSON
4.     private int WindowWidth;
5.     private List<Colours> colours; // The colours as a list of the colours class
6.     // Getter and setter methods
7.     public int getWindowHeight() {
8.         return WindowHeight;
9.     }
10.    public void setWindowHeight(int WindowHeight) {
11.        this.WindowHeight = WindowHeight;
12.    }
13.    }
14.    public int getWindowWidth() {
15.        return WindowWidth;
16.    }
17.    }
18.    public void setWindowWidth(int WindowWidth) {
19.        this.WindowWidth = WindowWidth;
20.    }
21.    }
22.    public List<Colours> getColours() {
23.        return colours;
24.    }
25.    }
26.    public void setColours(List<Colours> colours) {
27.        this.colours = colours;
28.    }
29.    }
30. }

```

C2.3 Organism.java

```
1. public class Organism {
2.     private int organismId; // Declaring each variable
3.     private Stats stats;
4.     private String currentTerrain;
5.     private int xCoordinate;
6.     private int yCoordinate;
7.     private int sex; // 1 is male, 0 is female
8.     private int fatherID;
9.     private int motherID;
10.    private boolean predator;
11.    // Getter and setter methods for each
12.    public int getOrganismId() { // Getter and setter methods for each variable
13.        return organismId;
14.    }
15.    public void setOrganismId(int organismId) {
16.        this.organismId = organismId;
17.    }
18.    public Stats getStats() {
19.        return stats;
20.    }
21.    public void setStats(Stats stats) {
22.        this.stats = stats;
23.    }
24.    public String getCurrentTerrain() {
25.        return currentTerrain;
26.    }
27.    public void setCurrentTerrain(String currentTerrain) {
28.        this.currentTerrain = currentTerrain;
29.    }
30.    public int getxCoordinate() {
31.        return xCoordinate;
32.    }
33.    public void setxCoordinate(int xCoordinate) {
34.        this.xCoordinate = xCoordinate;
35.    }
36.    public int getyCoordinate() {
37.        return yCoordinate;
38.    }
39.    public void setyCoordinate(int yCoordinate) {
40.        this.yCoordinate = yCoordinate;
41.    }
42.    public void setSex(int sex){
43.        this.sex = sex;
44.    }
45.    public int getSex() {
46.        return sex;
47.    }
48.    public int getFatherID() {
49.        return fatherID;
50.    }
51.    public void setFatherID(int fatherID) {
52.        this.fatherID = fatherID;
53.    }
54.    public int getMotherID() {
55.        return motherID;
56.    }
57.    public void setMotherID(int motherID) {
58.        this.motherID = motherID;
```

```

59.    }
60.
61.    public boolean isPredator() {
62.        return predator;
63.    }
64.
65.    public void setPredator(boolean predator) {
66.        this.predator = predator;
67.    }
68. }
```

C2.4 Settings.java

```

1. public class Settings {
2.     private double seed; // Initialising each variable
3.     private int currentYear;
4.     private int windowHeight;
5.     private int windowHeight;
6.     private boolean predatorPreyEnabled;
7.     private boolean seasonsEnabled;
8.     private String currentSeason;
9.     // Getter and setter for each
10.    public double getSeed() {
11.        return seed;
12.    }
13.
14.    public void setSeed(double seed) {
15.        this.seed = seed;
16.    }
17.
18.    public int getCurrentYear() {
19.        return currentYear;
20.    }
21.
22.    public void setCurrentYear(int currentYear) {
23.        this.currentYear = currentYear;
24.    }
25.
26.    public int getWindowHeight() {
27.        return windowHeight;
28.    }
29.
30.    public int getWindowWidth() {
31.        return windowHeight;
32.    }
33.
34.    public void setWindowHeight(int windowHeight) {
35.        this.windowHeight = windowHeight;
36.    }
```

```

37.     public void setWindowWidth(int windowHeight) {
38.         this.windowWidth = windowHeight;
39.     }
40.
41.     public boolean isSeasonsEnabled() {
42.         return seasonsEnabled;
43.     }
44.
45.     public boolean isPredatorPreyEnabled() {
46.         return predatorPreyEnabled;
47.     }
48.
49.
50.     public void setSeasonsEnabled(boolean seasonsEnabled) {
51.         this.seasonsEnabled = seasonsEnabled;
52.     }
53.     public void setPredatorPreyEnabled(boolean predatorPreyEnabled) {
54.         this.predatorPreyEnabled = predatorPreyEnabled;
55.     }
56.
57.     public String getCurrentSeason() {
58.         return currentSeason;
59.     }
60.
61.     public void setCurrentSeason(String season) {
62.         this.currentSeason = season;
63.     }
64. }
```

C2.5 Stats.java

```

1. import FoodTypes.FoodTypes;
2. import java.util.List;
3.
4. public class Stats {
5.     private double currentAwareness; // Initialising each variable
6.     private double organismsEncountered;
7.     private List<Double> awarenessHistory;
8.     private int age;
9.     private FoodTypes preferredFood;
10.    private String preferredFoodString;
11.    private FoodTypes eatenFood;
12.    private List<FoodTypes> recentFood;
13.    private double foodFactor;
14.    private double sizeFactor;
```

```

15.     private double size;
16.     private double speed;
17.     private double startSpeed;
18.     private int currentTemp;
19.     private int preferredTemp;
20.     private List<Integer> recentTemp;
21.     private double tempFactor;
22.     private String preferredTerrain;
23.     private String secondPreferredTerrain;
24.     private String thirdPreferredTerrain;
25.     private double health;
26.     private boolean dead;
27.     private int ageLastBred;
28.     private int generation;
29.     private int organismsMet;
30.     private int totalOrganismsMet;
31.     private String tsb;
32.     private int lastAttack;
33.     // Getter and setter for each
34.     public double getCurrentAwareness(){
35.         return currentAwareness;
36.     }
37.     public void setCurrentAwareness(double currentAwareness){
38.         this.currentAwareness = currentAwareness;
39.     }
40.     public double getOrganismsEncountered(){
41.         return organismsEncountered;
42.     }
43.     public void setOrganismsEncountered(double organismsEncountered){
44.         this.organismsEncountered = organismsEncountered;
45.     }
46.     public List<Double> getAwarenessHistory() {
47.         return awarenessHistory;
48.     }
49.     public void setAwarenessHistory(List<Double> awarenessHistory){
50.         this.awarenessHistory = awarenessHistory;
51.     }
52.     public int getAge() {
53.         return age;
54.     }
55.     public void setAge(int age) {
56.         this.age = age;
57.     }
58.     public FoodTypes getPreferredFood(){
59.         return preferredFood;
60.     }
61.     public void setPreferredFood(FoodTypes preferredFood){
62.         this.preferredFood = preferredFood;
63.     }
64.     public FoodTypes getEatenFood(){
65.         return eatenFood;
66.     }
67.     public void setEatenFood(FoodTypes eatenFood){
68.         this.eatenFood = eatenFood;
69.     }
70.     public List<FoodTypes> getRecentFood(){
71.         return recentFood;
72.     }
73.     public void setRecentFood(List<FoodTypes> recentFood){
74.         this.recentFood = recentFood;
75.     }
76.     public double getFoodFactor(){
77.         return foodFactor;
78.     }
79.     public void setFoodFactor(double foodFactor){
80.         this.foodFactor = foodFactor;
81.     }
82.     public double getSizeFactor(){
83.         return sizeFactor;
84.     }
85.     public void setSizeFactor(double sizeFactor){

```

```

86.         this.sizeFactor = sizeFactor;
87.     }
88.     public double getSize(){
89.         return size;
90.     }
91.     public void setSize(double size){
92.         this.size = size;
93.     }
94.     public double getSpeed(){
95.         return speed;
96.     }
97.     public void setSpeed(double speed){
98.         this.speed = speed;
99.     }
100.    public int getCurrentTemp(){
101.        return currentTemp;
102.    }
103.    public void setCurrentTemp(int currentTemp) {
104.        this.currentTemp = currentTemp;
105.    }
106.    public int getPreferredTemp(){
107.        return preferredTemp;
108.    }
109.    public void setPreferredTemp(int preferredTemp) {
110.        this.preferredTemp = preferredTemp;
111.    }
112.    public List<Integer> getRecentTemp(){
113.        return recentTemp;
114.    }
115.    public void setRecentTemp(List<Integer> recentTemp){
116.        this.recentTemp = recentTemp;
117.    }
118.    public double getTempFactor(){
119.        return tempFactor;
120.    }
121.    public void setTempFactor(double tempFactor) {
122.        this.tempFactor = tempFactor;
123.    }
124.
125.    public String getPreferredTerrain() {
126.        return preferredTerrain;
127.    }
128.
129.    public void setPreferredTerrain(String preferredTerrain) {
130.        this.preferredTerrain = preferredTerrain;
131.    }
132.
133.    public String getSecondPreferredTerrain() {
134.        return secondPreferredTerrain;
135.    }
136.
137.    public void setSecondPreferredTerrain(String secondPreferredTerrain) {
138.        this.secondPreferredTerrain = secondPreferredTerrain;
139.    }
140.
141.    public String getThirdPreferredTerrain() {
142.        return thirdPreferredTerrain;
143.    }
144.
145.    public void setThirdPreferredTerrain(String thirdPreferredTerrain) {
146.        this.thirdPreferredTerrain = thirdPreferredTerrain;
147.    }
148.
149.    public double getHealth() {
150.        return health;
151.    }
152.
153.    public void setHealth(double health) {
154.        this.health = health;
155.    }
156.

```

```
157.     public boolean isDead() {
158.         return dead;
159.     }
160.
161.     public void setDead(boolean dead) {
162.         this.dead = dead;
163.     }
164.
165.     public int getAgeLastBred() {
166.         return ageLastBred;
167.     }
168.
169.     public void setAgeLastBred(int ageLastBred) {
170.         this.ageLastBred = ageLastBred;
171.     }
172.
173.     public String getPreferredFoodString() {
174.         return preferredFoodString;
175.     }
176.
177.     public void setPreferredFoodString(String preferredFoodString) {
178.         this.preferredFoodString = preferredFoodString;
179.     }
180.
181.     public int getGeneration() {
182.         return generation;
183.     }
184.
185.     public double getStartSpeed() {
186.         return startSpeed;
187.     }
188.
189.     public void setStartSpeed(double startSpeed) {
190.         this.startSpeed = startSpeed;
191.     }
192.
193.     public int getOrganismsMet() {
194.         return organismsMet;
195.     }
196.
197.     public void setOrganismsMet(int organismsMet) {
198.         this.organismsMet = organismsMet;
199.     }
200.
201.     public String getTSB() {
202.         return tsb;
203.     }
204.
205.     public void setTSB(String TSB) {
206.         this.tsb = TSB;
207.     }
208.
209.     public int getTotalOrganismsMet() {
210.         return totalOrganismsMet;
211.     }
212.
213.     public void setTotalOrganismsMet(int totalOrganismsMet) {
214.         this.totalOrganismsMet = totalOrganismsMet;
215.     }
216.
217.     public int getLastAttack() {
218.         return lastAttack;
219.     }
220.
221.     public void setLastAttack(int lastAttack) {
222.         this.lastAttack = lastAttack;
223.     }
224. }
```

This section now contains .form files, which are created by IntelliJ forms creator.

C2.6 LoadSimulation.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="LoadSimulation">
3.   <grid id="27dc6" binding="panel1" layout-manager="GridLayoutManager" row-count="5" column-
count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
4.     <margin top="0" left="0" bottom="0" right="0"/>
5.     <constraints>
6.       <xy x="20" y="20" width="504" height="529"/>
7.     </constraints>
8.     <properties>
9.       <background color="-10328731"/>
10.      <preferredSize width="500" height="500"/>
11.    </properties>
12.    <border type="none"/>
13.    <children>
14.      <component id="708a7" class="javax.swing.JLabel">
15.        <constraints>
16.          <grid row="0" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false"/>
17.        </constraints>
18.        <properties>
19.          <foreground color="-16777216"/>
20.          <horizontalAlignment value="11"/>
21.          <text value="Load Simulation"/>
22.        </properties>
23.      </component>
24.      <component id="59c" class="javax.swing.JButton" binding="loadSaveButton" default-
binding="true">
25.        <constraints>
26.          <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false">
27.            <preferred-size width="230" height="-1"/>
28.          </grid>
29.        </constraints>
30.        <properties>
31.          <background color="-9210250"/>
32.          <borderPainted value="false"/>
33.          <focusPainted value="false"/>
34.          <text value="Load Save"/>
35.        </properties>
36.      </component>
37.      <component id="71b65" class="javax.swing.JButton" binding="cancelButton" default-
binding="true">
38.        <constraints>
39.          <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="4" fill="0" indent="0" use-parent-layout="false">
40.            <preferred-size width="230" height="-1"/>
41.          </grid>
42.        </constraints>
43.        <properties>
44.          <background color="-9210250"/>
45.          <borderPainted value="false"/>
46.          <focusPainted value="false"/>
47.          <text value="Cancel"/>
48.        </properties>
49.      </component>
50.      <component id="812fc" class="javax.swing.JList" binding="list1" default-binding="true">
51.        <constraints>
52.          <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="6" hsize-policy="2" anchor="0" fill="0" indent="0" use-parent-layout="false">
53.            <preferred-size width="450" height="400"/>
54.          </grid>
55.        </constraints>
56.        <properties>
57.          <background color="-6709859"/>
58.          <selectionBackground color="-6709859"/>
59.          <selectionForeground color="-393217"/>
```

```

60.      </properties>
61.    </component>
62.    <component id="224b9" class="javax.swing.JButton" binding="deleteButton" default-
binding="true">
63.      <constraints>
64.        <grid row="2" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3"
anchor="8" fill="0" indent="0" use-parent-layout="false">
65.          <preferred-size width="230" height="-1"/>
66.        </grid>
67.      </constraints>
68.      <properties>
69.        <background color="-9210250"/>
70.        <borderPainted value="false"/>
71.        <text value="Delete"/>
72.      </properties>
73.    </component>
74.    <component id="ded76" class="javax.swing.JButton" binding="renameButton" default-
binding="true">
75.      <constraints>
76.        <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3"
anchor="4" fill="0" indent="0" use-parent-layout="false">
77.          <preferred-size width="230" height="-1"/>
78.        </grid>
79.      </constraints>
80.      <properties>
81.        <background color="-9210250"/>
82.        <borderPainted value="false"/>
83.        <text value="Rename"/>
84.      </properties>
85.    </component>
86.    <vspacer id="71bd7">
87.      <constraints>
88.        <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-policy="1"
anchor="0" fill="2" indent="0" use-parent-layout="false"/>
89.      </constraints>
90.    </vspacer>
91.  </children>
92. </grid>
93. </form>

```

C2.7 MainMenuOptions.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="LoadSimulation">
3.   <grid id="27dc6" binding="panel1" layout-manager="GridLayoutManager" row-count="5" column-
count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
4.     <margin top="0" left="0" bottom="0" right="0"/>
5.     <constraints>
6.       <xy x="20" y="20" width="504" height="529"/>
7.     </constraints>
8.     <properties>
9.       <background color="-10328731"/>
10.      <preferredSize width="500" height="500"/>
11.    </properties>
12.    <border type="none"/>
13.    <children>
14.      <component id="708a7" class="javax.swing.JLabel">
15.        <constraints>
16.          <grid row="0" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false"/>
17.        </constraints>
18.        <properties>
19.          <foreground color="-16777216"/>
20.          <horizontalAlignment value="11"/>
21.          <text value="Load Simulation"/>
22.        </properties>
23.      </component>
24.      <component id="59c" class="javax.swing.JButton" binding="loadSaveButton" default-
binding="true">
25.        <constraints>
26.          <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false">
27.            <preferred-size width="230" height="-1"/>
28.          </grid>
29.        </constraints>
30.        <properties>
31.          <background color="-9210250"/>
32.          <borderPainted value="false"/>
33.          <focusPainted value="false"/>
34.          <text value="Load Save"/>
35.        </properties>
36.      </component>
37.      <component id="71b65" class="javax.swing.JButton" binding="cancelButton" default-
binding="true">
38.        <constraints>
39.          <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="4" fill="0" indent="0" use-parent-layout="false">
40.            <preferred-size width="230" height="-1"/>
41.          </grid>
42.        </constraints>
43.        <properties>
44.          <background color="-9210250"/>
45.          <borderPainted value="false"/>
46.          <focusPainted value="false"/>
47.          <text value="Cancel"/>
48.        </properties>
49.      </component>
50.      <component id="812fc" class="javax.swing.JList" binding="list1" default-binding="true">
51.        <constraints>
52.          <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="6" hsize-policy="2" anchor="0" fill="0" indent="0" use-parent-layout="false">
53.            <preferred-size width="450" height="400"/>
54.          </grid>
55.        </constraints>
56.        <properties>
57.          <background color="-6709859"/>
```

```

58.          <selectionBackground color="-6709859"/>
59.          <selectionForeground color="-393217"/>
60.        </properties>
61.      </component>
62.      <component id="224b9" class="javax.swing.JButton" binding="deleteButton" default-
binding="true">
63.        <constraints>
64.          <grid row="2" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3"
anchor="8" fill="0" indent="0" use-parent-layout="false">
65.            <preferred-size width="230" height="-1"/>
66.          </grid>
67.        </constraints>
68.        <properties>
69.          <background color="-9210250"/>
70.          <borderPainted value="false"/>
71.          <text value="Delete"/>
72.        </properties>
73.      </component>
74.      <component id="ded76" class="javax.swing.JButton" binding="renameButton" default-
binding="true">
75.        <constraints>
76.          <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3"
anchor="4" fill="0" indent="0" use-parent-layout="false">
77.            <preferred-size width="230" height="-1"/>
78.          </grid>
79.        </constraints>
80.        <properties>
81.          <background color="-9210250"/>
82.          <borderPainted value="false"/>
83.          <text value="Rename"/>
84.        </properties>
85.      </component>
86.      <vspacer id="71bd7">
87.        <constraints>
88.          <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-policy="1"
anchor="0" fill="2" indent="0" use-parent-layout="false"/>
89.        </constraints>
90.      </vspacer>
91.    </children>
92.  </grid>
93. </form>

```

C2.8 RenameFile.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="RenameFile">
3.   <grid id="27dc6" layout-manager="GridLayoutManager" row-count="1" column-count="1" same-size-
horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
4.     <margin top="0" left="0" bottom="0" right="0"/>
5.     <constraints>
6.       <xy x="20" y="20" width="500" height="400"/>
7.     </constraints>
8.     <properties>
9.       <background color="-10328731"/>
10.    </properties>
11.    <border type="none"/>
12.    <children>
13.      <grid id="63516" binding="panel1" layout-manager="GridLayoutManager" row-count="5"
column-count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
14.        <margin top="0" left="0" bottom="0" right="0"/>
15.        <constraints>
16.          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3"
anchor="0" fill="0" indent="0" use-parent-layout="false"/>
17.        </constraints>
18.        <properties>
19.          <background color="-10328731"/>
20.          <maximumSize width="300" height="175"/>
21.          <minimumSize width="150" height="100"/>
22.          <preferredSize width="300" height="175"/>
23.        </properties>
24.        <border type="none"/>
25.        <children>
26.          <component id="46e3d" class="javax.swing.JLabel">
27.            <constraints>
28.              <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="1" anchor="1" fill="0" indent="0" use-parent-layout="false">
29.                <preferred-size width="100" height="-1"/>
30.              </grid>
31.            </constraints>
32.            <properties>
33.              <foreground color="-16777216"/>
34.              <horizontalAlignment value="10"/>
35.              <horizontalTextPosition value="0"/>
36.              <text value="Rename File"/>
37.            </properties>
38.          </component>
39.          <component id="b2f08" class="javax.swing.JTextField" binding="renameNameTextField"
default-binding="true">
40.            <constraints>
41.              <grid row="2" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="3" anchor="0" fill="0" indent="0" use-parent-layout="false">
42.                <preferred-size width="250" height="-1"/>
43.              </grid>
44.            </constraints>
45.            <properties>
46.              <background color="-6709859"/>
47.              <caretColor color="-8881285"/>
48.              <text value="Rename Name"/>
49.            </properties>
50.          </component>
51.          <component id="86e64" class="javax.swing.JButton" binding="cancelButton" default-
binding="true">
52.            <constraints>
53.              <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="7" anchor="4" fill="0" indent="0" use-parent-layout="false">
54.                <preferred-size width="125" height="25"/>
55.              </grid>
56.            </constraints>
57.            <properties>
```

```

58.          <background color="-9210250"/>
59.          <borderPainted value="false"/>
60.          <margin top="0" left="1" bottom="0" right="0"/>
61.          <text value="Cancel"/>
62.      </properties>
63.  </component>
64.  <component id="f9798" class="javax.swing.JButton" binding="renameFileButton">
65.      <constraints>
66.          <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="7" anchor="8" fill="0" indent="0" use-parent-layout="false">
67.              <preferred-size width="125" height="-1"/>
68.          </grid>
69.      </constraints>
70.      <properties>
71.          <background color="-9210250"/>
72.          <borderPainted value="false"/>
73.          <text value="Rename"/>
74.      </properties>
75.  </component>
76.  <vspacer id="493b7">
77.      <constraints>
78.          <grid row="4" column="1" row-span="1" col-span="1" vsize-policy="6" hsize-
policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
79.      </constraints>
80.  </vspacer>
81.  <vspacer id="1f9e">
82.      <constraints>
83.          <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-
policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
84.      </constraints>
85.  </vspacer>
86.  <vspacer id="e5b5d">
87.      <constraints>
88.          <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="7" hsize-
policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
89.      </constraints>
90.  </vspacer>
91.  </children>
92.  </grid>
93.  </children>
94. </grid>
95. </form>

```

C2.9 SaveSimulation.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="SaveSimulation">
3.   <grid id="27dc6" binding="panel1" layout-manager="GridLayoutManager" row-count="5" column-
count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
4.     <margin top="0" left="0" bottom="0" right="0"/>
5.     <constraints>
6.       <xy x="20" y="20" width="500" height="400"/>
7.     </constraints>
8.     <properties>
9.       <background color="-10328731"/>
10.      <maximumSize width="300" height="175"/>
11.      <minimumSize width="150" height="100"/>
12.      <preferredSize width="300" height="175"/>
13.    </properties>
14.    <border type="none"/>
15.    <children>
16.      <component id="740a8" class="javax.swing.JLabel">
17.        <constraints>
18.          <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-policy="1"
anchor="1" fill="0" indent="0" use-parent-layout="false">
19.            <preferred-size width="100" height="-1"/>
20.          </grid>
21.        </constraints>
22.        <properties>
23.          <foreground color="-16777216"/>
24.          <horizontalAlignment value="10"/>
25.          <horizontalTextPosition value="0"/>
26.          <text value="Save Simulation"/>
27.        </properties>
28.      </component>
29.      <component id="e79e2" class="javax.swing.JTextField" binding="saveNameTextField" default-
binding="true">
30.        <constraints>
31.          <grid row="2" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-policy="3"
anchor="0" fill="0" indent="0" use-parent-layout="false">
32.            <preferred-size width="250" height="-1"/>
33.          </grid>
34.        </constraints>
35.        <properties>
36.          <background color="-6709859"/>
37.          <caretColor color="-8881285"/>
38.          <text value="Save Name"/>
39.        </properties>
40.      </component>
41.      <component id="9c0c0" class="javax.swing.JButton" binding="cancelButton" default-
binding="true">
42.        <constraints>
43.          <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="7"
anchor="4" fill="0" indent="0" use-parent-layout="false">
44.            <preferred-size width="125" height="25"/>
45.          </grid>
46.        </constraints>
47.        <properties>
48.          <background color="-9210250"/>
49.          <borderPainted value="false"/>
50.          <margin top="0" left="1" bottom="0" right="0"/>
51.          <text value="Cancel"/>
52.        </properties>
53.      </component>
54.      <component id="4ee10" class="javax.swing.JButton" binding="saveSimulationButton">
55.        <constraints>
56.          <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="7"
anchor="8" fill="0" indent="0" use-parent-layout="false">
57.            <preferred-size width="125" height="-1"/>
58.          </grid>
59.        </constraints>
```

```
60.      <properties>
61.        <background color="-9210250"/>
62.        <borderPainted value="false"/>
63.        <text value="Save Simulation"/>
64.      </properties>
65.    </component>
66.    <vspacer id="f8dc1">
67.      <constraints>
68.        <grid row="4" column="1" row-span="1" col-span="1" vsize-policy="6" hsize-policy="1"
anchor="0" fill="2" indent="0" use-parent-layout="false"/>
69.      </constraints>
70.    </vspacer>
71.    <vspacer id="7b903">
72.      <constraints>
73.        <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-policy="1"
anchor="0" fill="2" indent="0" use-parent-layout="false"/>
74.      </constraints>
75.    </vspacer>
76.    <vspacer id="647df">
77.      <constraints>
78.        <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="7" hsize-policy="1"
anchor="0" fill="2" indent="0" use-parent-layout="false"/>
79.      </constraints>
80.    </vspacer>
81.  </children>
82. </grid>
83. </form>
```

C2.10 SideMenu.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="SideMenu">
3.   <scrollpane id="801e3">
4.     <constraints>
5.       <xy x="20" y="20" width="268" height="728"/>
6.     </constraints>
7.     <properties>
8.       <autoscrolls value="false"/>
9.       <doubleBuffered value="true"/>
10.      <horizontalScrollBarPolicy value="32"/>
11.    </properties>
12.    <border type="none"/>
13.    <children>
14.      <grid id="27dc6" binding="SideMenu" layout-manager="GridLayoutManager" row-count="16"
column-count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
15.        <margin top="0" left="0" bottom="0" right="0"/>
16.        <constraints/>
17.        <properties>
18.          <autoscrolls value="true"/>
19.          <background color="-10328731"/>
20.          <preferredSize width="250" height="24"/>
21.        </properties>
22.        <border type="none"/>
23.        <children>
24.          <component id="2e008" class="javax.swing.JLabel">
25.            <constraints>
26.              <grid row="0" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false"/>
27.            </constraints>
28.            <properties>
29.              <foreground color="-16777216"/>
30.              <text value="EvSim"/>
31.            </properties>
32.          </component>
33.          <component id="3f146" class="javax.swing.JSeparator">
34.            <constraints>
35.              <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="1" hsize-
policy="6" anchor="1" fill="1" indent="0" use-parent-layout="false"/>
36.            </constraints>
37.            <properties>
38.              <background color="-16777216"/>
39.              <foreground color="-16777216"/>
40.            </properties>
41.          </component>
42.          <component id="9557c" class="javax.swing.JLabel" binding="currentGameSpeedLabel">
43.            <constraints>
44.              <grid row="2" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
45.            </constraints>
46.            <properties>
47.              <foreground color="-1"/>
48.              <text value="Current Game Speed:"/>
49.            </properties>
50.          </component>
51.          <component id="b5aa8" class="javax.swing.JButton" binding="speedUpButton" default-
binding="true">
52.            <constraints>
53.              <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="6" anchor="8" fill="0" indent="0" use-parent-layout="false">
54.                <preferred-size width="115" height="-1"/>
55.              </grid>
56.            </constraints>
57.            <properties>
58.              <background color="-9210250"/>
59.              <borderPainted value="false"/>
60.              <focusPainted value="false"/>
61.              <text value="Speed Up"/>
```

```

62.          </properties>
63.      </component>
64.      <component id="4ad36" class="javax.swing.JButton" binding="slowDownButton" default-
binding="true">
65.          <constraints>
66.              <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="6" anchor="4" fill="0" indent="0" use-parent-layout="false">
67.                  <preferred-size width="115" height="-1"/>
68.              </grid>
69.          </constraints>
70.          <properties>
71.              <background color="-9210250"/>
72.              <borderPainted value="false"/>
73.              <focusPainted value="false"/>
74.              <text value="Slow Down"/>
75.          </properties>
76.      </component>
77.      <component id="9f766" class="javax.swing.JSeparator">
78.          <constraints>
79.              <grid row="6" column="0" row-span="1" col-span="2" vsize-policy="1" hsize-
policy="6" anchor="1" fill="1" indent="0" use-parent-layout="false"/>
80.          </constraints>
81.          <properties>
82.              <background color="-16777216"/>
83.              <foreground color="-16777216"/>
84.          </properties>
85.      </component>
86.      <component id="6f6f9" class="javax.swing.JLabel">
87.          <constraints>
88.              <grid row="7" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false"/>
89.          </constraints>
90.          <properties>
91.              <foreground color="-16777216"/>
92.              <text value="Organism Information"/>
93.          </properties>
94.      </component>
95.      <component id="9455f" class="javax.swing.JButton" binding="pauseGameButton" default-
binding="true">
96.          <constraints>
97.              <grid row="5" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="6" anchor="0" fill="0" indent="0" use-parent-layout="false">
98.                  <preferred-size width="230" height="-1"/>
99.              </grid>
100.          </constraints>
101.          <properties>
102.              <background color="-9210250"/>
103.              <borderPainted value="false"/>
104.              <focusPainted value="false"/>
105.              <text value="Pause Game"/>
106.          </properties>
107.      </component>
108.      <component id="dc4a4" class="javax.swing.JLabel" binding="currentYearLabel">
109.          <constraints>
110.              <grid row="4" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
111.          </constraints>
112.          <properties>
113.              <foreground color="-1"/>
114.              <text value="Current Year: />
115.          </properties>
116.      </component>
117.      <component id="c29a5" class="javax.swing.JButton" binding="loadPreviousSaveButton"
default-binding="true">
118.          <constraints>
119.              <grid row="12" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="6" anchor="0" fill="0" indent="0" use-parent-layout="false">
120.                  <preferred-size width="230" height="-1"/>
121.              </grid>
122.          </constraints>
123.          <properties>

```

```

124.          <background color="-9210250"/>
125.          <borderPainted value="false"/>
126.          <focusPainted value="false"/>
127.          <text value="Load Previous Save"/>
128.      </properties>
129.  </component>
130.  <component id="f579f" class="javax.swing.JButton" binding="exitToMainMenuButton"
default-binding="true">
131.      <constraints>
132.          <grid row="13" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="6" anchor="0" fill="0" indent="0" use-parent-layout="false">
133.              <preferred-size width="230" height="-1"/>
134.          </grid>
135.      </constraints>
136.      <properties>
137.          <background color="-9210250"/>
138.          <borderPainted value="false"/>
139.          <focusPainted value="false"/>
140.          <text value="Exit to Main Menu"/>
141.      </properties>
142.  </component>
143.  <component id="eb321" class="javax.swing.JButton" binding="saveSimulationButton"
default-binding="true">
144.      <constraints>
145.          <grid row="11" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="6" anchor="0" fill="0" indent="0" use-parent-layout="false">
146.              <preferred-size width="230" height="-1"/>
147.          </grid>
148.      </constraints>
149.      <properties>
150.          <background color="-9210250"/>
151.          <borderPainted value="false"/>
152.          <focusPainted value="false"/>
153.          <text value="Save Simulation"/>
154.      </properties>
155.  </component>
156.  <tabbedPane id="a0dec" binding="organismInfoTab">
157.      <constraints>
158.          <grid row="8" column="0" row-span="1" col-span="2" vsize-policy="3" hsize-
policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false">
159.              <preferred-size width="200" height="400"/>
160.          </grid>
161.      </constraints>
162.      <properties>
163.          <background color="-10328731"/>
164.          <foreground color="-393217"/>
165.          <inheritsPopupMenu value="false"/>
166.          <opaque value="false"/>
167.          <tabLayoutPolicy value="0"/>
168.      </properties>
169.      <border type="none"/>
170.      <children>
171.          <grid id="3cd4a" layout-manager="GridLayoutManager" row-count="1" column-
count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
172.              <margin top="0" left="0" bottom="0" right="0"/>
173.          <constraints>
174.              <tabbedPane title="Global View"/>
175.          </constraints>
176.          <properties>
177.              <focusable value="false"/>
178.              <preferredSize width="208" height="300"/>
179.          </properties>
180.          <border type="none"/>
181.          <children>
182.              <scrollPane id="3f2a3">
183.                  <constraints>
184.                      <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="7"
hsize-policy="7" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
185.                  </constraints>
186.                  <properties>
187.                      <background color="-10328731"/>

```

```

188.          <visible value="true"/>
189.      </properties>
190.      <border type="none"/>
191.      <children>
192.          <grid id="dcb00" binding="organismInformationPanel" layout-
manager="GridLayoutManager" row-count="10" column-count="1" same-size-horizontally="false" same-
size-vertically="false" hgap="-1" vgap="-1">
193.              <margin top="0" left="0" bottom="0" right="0"/>
194.              <constraints/>
195.              <properties>
196.                  <autoscrolls value="true"/>
197.                  <background color="-10328731"/>
198.                  <enabled value="true"/>
199.                  <focusable value="true"/>
200.                  <visible value="true"/>
201.              </properties>
202.              <border type="none"/>
203.              <children>
204.                  <component id="3afe2" class="javax.swing.JLabel"
binding="organismCountLabel">
205.                      <constraints>
206.                          <grid row="0" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
207.                      </constraints>
208.                      <properties>
209.                          <foreground color="-1"/>
210.                          <text value="Organism Count:"/>
211.                      </properties>
212.                  </component>
213.                  <component id="45592" class="javax.swing.JLabel"
binding="oldestOrganismLabel">
214.                      <constraints>
215.                          <grid row="3" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
216.                      </constraints>
217.                      <properties>
218.                          <foreground color="-1"/>
219.                          <text value="Oldest Organism:"/>
220.                      </properties>
221.                  </component>
222.                  <component id="f9843" class="javax.swing.JLabel"
binding="healthiestOrganismLabel">
223.                      <constraints>
224.                          <grid row="4" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
225.                      </constraints>
226.                      <properties>
227.                          <foreground color="-1"/>
228.                          <text value="Healthiest Organism:"/>
229.                      </properties>
230.                  </component>
231.                  <component id="68b06" class="javax.swing.JLabel"
binding="fastestOrganismLabel">
232.                      <constraints>
233.                          <grid row="5" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
234.                      </constraints>
235.                      <properties>
236.                          <foreground color="-1"/>
237.                          <text value="Fastest Organism:"/>
238.                      </properties>
239.                  </component>
240.                  <component id="153f6" class="javax.swing.JLabel"
binding="mostPopulatedTerrainLabel">
241.                      <constraints>
242.                          <grid row="7" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
243.                      </constraints>
244.                      <properties>
245.                          <foreground color="-1"/>
246.                          <text value="Most Populated Terrain:"/>

```

```

247.                     </properties>
248.                 </component>
249.             <component id="78e37" class="javax.swing.JLabel"
binding="predatorCountLabel">
250.                     <constraints>
251.                         <grid row="1" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
252.                     </constraints>
253.                 <properties>
254.                     <enabled value="true"/>
255.                     <foreground color="-1"/>
256.                     <text value="Predator Count:"/>
257.                 </properties>
258.             </component>
259.             <component id="e068" class="javax.swing.JLabel" binding="mostAware">
260.                     <constraints>
261.                         <grid row="2" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
262.                     </constraints>
263.                 <properties>
264.                     <foreground color="-1"/>
265.                     <text value="Prey Count:"/>
266.                 </properties>
267.             </component>
268.             <component id="2c88c" class="javax.swing.JLabel"
binding="averageTemperatureLabel">
269.                     <constraints>
270.                         <grid row="6" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
271.                     </constraints>
272.                 <properties>
273.                     <foreground color="-1"/>
274.                     <text value="Average Temperature:"/>
275.                 </properties>
276.             </component>
277.             <component id="2553e" class="javax.swing.JLabel"
binding="leastPopulatedTerrainLabel">
278.                     <constraints>
279.                         <grid row="8" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
280.                     </constraints>
281.                 <properties>
282.                     <foreground color="-1"/>
283.                     <text value="Least Populated Terrain:"/>
284.                 </properties>
285.             </component>
286.             <component id="698d3" class="javax.swing.JLabel"
binding="newOrganismsSinceYearLabel">
287.                     <constraints>
288.                         <grid row="9" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="1" use-parent-layout="false"/>
289.                     </constraints>
290.                 <properties>
291.                     <foreground color="-1"/>
292.                     <text value="New Organisms Since Year 0: "/>
293.                 </properties>
294.             </component>
295.         </children>
296.     </grid>
297. </children>
298. </scrollpane>
299. </children>
300. </grid>
301. <grid id="68be3" layout-manager="GridLayoutManager" row-count="2" column-
count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
302.     <margin top="0" left="0" bottom="0" right="0"/>
303.     <constraints>
304.         <tabbedpane title="Key"/>
305.     </constraints>
306.     <properties>
307.         <background color="-10328731"/>

```

```

308.          </properties>
309.          <border type="none"/>
310.          <children>
311.              <component id="85ba0" class="javax.swing.JLabel" binding="KeyLabel">
312.                  <constraints>
313.                      <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0"
hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
314.                  </constraints>
315.                  <properties>
316.                      <text value="" />
317.                  </properties>
318.              </component>
319.              <vspacer id="256f9">
320.                  <constraints>
321.                      <grid row="1" column="0" row-span="1" col-span="1" vsize-policy="6"
hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
322.                  </constraints>
323.              </vspacer>
324.          </children>
325.      </grid>
326.      <grid id="ade6" layout-manager="GridLayoutManager" row-count="4" column-
count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
327.          <margin top="0" left="0" bottom="0" right="0" />
328.          <constraints>
329.              <tabbedpane title="Find Organism"/>
330.          </constraints>
331.          <properties>
332.              <background color="-10328731" />
333.          </properties>
334.          <border type="none"/>
335.          <children>
336.              <component id="8e715" class="javax.swing.JLabel">
337.                  <constraints>
338.                      <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0"
hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
339.                  </constraints>
340.                  <properties>
341.                      <foreground color="-393217" />
342.                      <text value="Enter Organism ID to Find" />
343.                  </properties>
344.              </component>
345.              <vspacer id="502b9">
346.                  <constraints>
347.                      <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="6"
hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
348.                  </constraints>
349.              </vspacer>
350.              <component id="e063c" class="javax.swing.JTextField"
binding="organismToFind">
351.                  <constraints>
352.                      <grid row="1" column="0" row-span="1" col-span="1" vsize-policy="0"
hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-layout="false">
353.                          <preferred-size width="150" height="-1" />
354.                      </grid>
355.                  </constraints>
356.                  <properties>
357.                      <background color="-6709859" />
358.                  </properties>
359.              </component>
360.              <component id="a146" class="javax.swing.JButton" binding="findButton"
default-binding="true">
361.                  <constraints>
362.                      <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="0"
hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
363.                  </constraints>
364.                  <properties>
365.                      <background color="-9210250" />
366.                      <borderPainted value="false" />
367.                      <contentAreaFilled value="true" />
368.                      <focusPainted value="true" />
369.                      <text value="Find" />

```

```

370.          </properties>
371.          </component>
372.          </children>
373.          </grid>
374.          </children>
375.        </tabbedpane>
376.        <component id="3241c" class="javax.swing.JCheckBox"
binding="showDeadOrganismsCheckBox" default-binding="true">
377.          <constraints>
378.            <grid row="10" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="3" anchor="8" fill="0" indent="2" use-parent-layout="false">
379.              <preferred-size width="230" height="-1"/>
380.            </grid>
381.          </constraints>
382.          <properties>
383.            <background color="-10328731"/>
384.            <contentAreaFilled value="false"/>
385.            <selected value="false"/>
386.            <text value="Show Dead Organisms"/>
387.          </properties>
388.        </component>
389.        <component id="8b13f" class="javax.swing.JCheckBox" binding="showConsoleCheckbox">
390.          <constraints>
391.            <grid row="9" column="0" row-span="1" col-span="2" vsize-policy="0" hsize-
policy="3" anchor="8" fill="0" indent="2" use-parent-layout="false">
392.              <preferred-size width="230" height="-1"/>
393.            </grid>
394.          </constraints>
395.          <properties>
396.            <background color="-10328731"/>
397.            <contentAreaFilled value="false"/>
398.            <selected value="false"/>
399.            <text value="Show Console"/>
400.          </properties>
401.        </component>
402.        <vspacer id="b3d8c">
403.          <constraints>
404.            <grid row="15" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-
policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
405.          </constraints>
406.        </vspacer>
407.        <component id="483da" class="javax.swing.JSeparator">
408.          <constraints>
409.            <grid row="14" column="0" row-span="1" col-span="2" vsize-policy="1" hsize-
policy="6" anchor="1" fill="1" indent="0" use-parent-layout="false"/>
410.          </constraints>
411.          <properties>
412.            <background color="-16777216"/>
413.            <foreground color="-16777216"/>
414.          </properties>
415.        </component>
416.      </children>
417.    </grid>
418.    </children>
419.  </scrollpane>
420. </form>

```

C2.11 StartGameMenu.form

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="StartGameMenu">
3.   <grid id="27dc6" binding="panel1" default-binding="true" layout-manager="GridLayoutManager"
row-count="1" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-
1" vgap="-1">
4.     <margin top="0" left="0" bottom="0" right="0"/>
5.     <constraints>
6.       <xy x="20" y="20" width="532" height="400"/>
7.     </constraints>
8.     <properties>
9.       <background color="-10328731"/>
10.      <opaque value="true"/>
11.    </properties>
12.    <clientProperties>
13.      <html.disable class="java.lang.Boolean" value="true"/>
14.    </clientProperties>
15.    <border type="none"/>
16.    <children>
17.      <grid id="ff27" layout-manager="GridLayoutManager" row-count="9" column-count="2" same-
size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
18.        <margin top="0" left="0" bottom="0" right="0"/>
19.        <constraints>
20.          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3"
anchor="0" fill="3" indent="0" use-parent-layout="false"/>
21.        </constraints>
22.        <properties>
23.          <background color="-10328731"/>
24.        </properties>
25.        <border type="line" title="Game Colours" title-justification="1" title-position="2">
26.          <font name="Agency FB"/>
27.          <color color="-16777216"/>
28.        </border>
29.        <children>
30.          <component id="2e95a" class="javax.swing.JTextField" binding="seedTextField"
default-binding="true">
31.            <constraints>
32.              <grid row="1" column="0" row-span="1" col-span="2" vsize-policy="4" hsize-
policy="6" anchor="8" fill="1" indent="0" use-parent-layout="false">
33.                <preferred-size width="150" height="-1"/>
```

```

34.          </grid>
35.      </constraints>
36.      <properties>
37.          <background color="-8355712"/>
38.          <text value="0.0"/>
39.          <toolTipText value="Input a custom seed here"/>
40.      </properties>
41.  </component>
42.  <component id="59726" class="javax.swing.JTextField" binding="organismTextField">
43.      <constraints>
44.          <grid row="3" column="0" row-span="1" col-span="2" vsize-policy="4" hsize-
policy="6" anchor="8" fill="1" indent="0" use-parent-layout="false">
45.              <preferred-size width="150" height="-1"/>
46.          </grid>
47.      </constraints>
48.      <properties>
49.          <background color="-8355712"/>
50.          <text value="50"/>
51.          <toolTipText value="organismCount"/>
52.      </properties>
53.  </component>
54.  <component id="899cd" class="javax.swing.JCheckBox" binding="allowSeasonsCheckBox"
default-binding="true">
55.      <constraints>
56.          <grid row="4" column="0" row-span="1" col-span="2" vsize-policy="4" hsize-
policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
57.      </constraints>
58.      <properties>
59.          <background color="-10328731"/>
60.          <selected value="true"/>
61.          <text value="Allow seasons"/>
62.      </properties>
63.  </component>
64.  <component id="f4d01" class="javax.swing.JCheckBox"
binding="allowPredatorPreyOrganismsCheckBox" default-binding="true">
65.      <constraints>
66.          <grid row="5" column="0" row-span="1" col-span="2" vsize-policy="4" hsize-
policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
67.      </constraints>
68.      <properties>
69.          <background color="-10328731"/>
70.          <selected value="true"/>
71.          <text value="Allow predator/prey Organisms"/>
72.      </properties>
73.  </component>
74.  <component id="1026d" class="javax.swing.JLabel">
75.      <constraints>
76.          <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="4" hsize-
policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
77.      </constraints>
78.      <properties>
79.          <font/>
80.          <text value="Organisms to generate:"/>
81.      </properties>
82.  </component>
83.  <component id="1c654" class="javax.swing.JLabel">
84.      <constraints>
85.          <grid row="6" column="0" row-span="1" col-span="1" vsize-policy="4" hsize-
policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
86.      </constraints>
87.      <properties>
88.          <font/>
89.          <text value="Window Size:"/>
90.      </properties>
91.  </component>
92.  <component id="13f82" class="javax.swing.JLabel">
93.      <constraints>
94.          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="4" hsize-
policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
95.      </constraints>
96.      <properties>

```

```

97.          <font/>
98.          <text value="Map Seed:"/>
99.      </properties>
100.     </component>
101.     <component id="a1971" class="javax.swing.JButton" binding="previewButton">
102.         <constraints>
103.             <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
104.         </constraints>
105.         <properties>
106.             <background color="-9210250"/>
107.             <borderPainted value="false"/>
108.             <text value="Preview Map"/>
109.             <toolTipText value="Preview what the map looks like before heading in"/>
110.         </properties>
111.     </component>
112.     <grid id="ca8ae" layout-manager="GridLayoutManager" row-count="1" column-count="2"
same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
113.         <margin top="0" left="0" bottom="0" right="0"/>
114.         <constraints>
115.             <grid row="8" column="0" row-span="1" col-span="2" vsize-policy="3" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false">
116.                 <preferred-size width="450" height="50"/>
117.             </grid>
118.         </constraints>
119.         <properties>
120.             <background color="-10328731"/>
121.         </properties>
122.         <border type="none"/>
123.         <children>
124.             <component id="f2485" class="javax.swing.JButton" binding="startGameButton"
default-binding="true">
125.                 <constraints>
126.                     <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false">
127.                         <preferred-size width="250" height="-1"/>
128.                     </grid>
129.                 </constraints>
130.                 <properties>
131.                     <background color="-9210250"/>
132.                     <borderPainted value="false"/>
133.                     <focusPainted value="false"/>
134.                     <text value="Start Game"/>
135.                     <toolTipText value="Starts the simulation"/>
136.                 </properties>
137.             </component>
138.             <component id="5711e" class="javax.swing.JButton" binding="backToHomeButton"
default-binding="true">
139.                 <constraints>
140.                     <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="5" anchor="0" fill="1" indent="0" use-parent-layout="false">
141.                         <preferred-size width="250" height="-1"/>
142.                     </grid>
143.                 </constraints>
144.                 <properties>
145.                     <background color="-9210250"/>
146.                     <borderPainted value="false"/>
147.                     <focusPainted value="false"/>
148.                     <text value="Back to Home"/>
149.                 </properties>
150.             </component>
151.         </children>
152.     </grid>
153.     <grid id="3d1f2" layout-manager="GridLayoutManager" row-count="1" column-count="3"
same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">
154.         <margin top="0" left="0" bottom="0" right="0"/>
155.         <constraints>
156.             <grid row="7" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-
policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false">
157.                 <preferred-size width="250" height="50"/>
158.             </grid>

```

```

159.          </constraints>
160.          <properties>
161.              <background color="-10328731"/>
162.          </properties>
163.          <border type="none"/>
164.          <children>
165.              <component id="86862" class="javax.swing.JTextField" binding="a1920TextField"
default-binding="true">
166.                  <constraints>
167.                      <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="4" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false">
168.                          <preferred-size width="75" height="25"/>
169.                      </grid>
170.                  </constraints>
171.                  <properties>
172.                      <background color="-8355712"/>
173.                      <caretColor color="-9210250"/>
174.                      <editable value="true"/>
175.                      <enabled value="true"/>
176.                      <horizontalAlignment value="0"/>
177.                      <selectedTextColor color="-2104859"/>
178.                      <text value="1920"/>
179.                  </properties>
180.              </component>
181.              <component id="d4eeb" class="javax.swing.JLabel">
182.                  <constraints>
183.                      <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="1" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false">
184.                          <preferred-size width="25" height="50"/>
185.                          <maximum-size width="10" height="-1"/>
186.                      </grid>
187.                  </constraints>
188.                  <properties>
189.                      <enabled value="true"/>
190.                      <text value="x"/>
191.                  </properties>
192.              </component>
193.              <component id="e5bfc" class="javax.swing.JTextField" binding="a1080TextField"
default-binding="true">
194.                  <constraints>
195.                      <grid row="0" column="2" row-span="1" col-span="1" vsize-policy="0" hsize-
policy="0" anchor="0" fill="0" indent="0" use-parent-layout="false">
196.                          <preferred-size width="75" height="25"/>
197.                      </grid>
198.                  </constraints>
199.                  <properties>
200.                      <background color="-8355712"/>
201.                      <caretColor color="-9210250"/>
202.                      <editable value="true"/>
203.                      <enabled value="true"/>
204.                      <horizontalAlignment value="0"/>
205.                      <text value="1080"/>
206.                      <toolTipText value="Height"/>
207.                  </properties>
208.              </component>
209.          <children>
210.      </grid>
211.      </children>
212.  </grid>
213. </children>
214. </grid>
215. </form>
216.

```

C3 Packages

C3.1 Terrains

For the terrains section, the individual terrain classes are uncommented due to C3.1.1 being the parent, therefore containing comments that cover all other files.

C3.1.1 TerrainAttributes.java

```
1. package Terrains;
2. import java.util.ArrayList;
3. import java.util.Random;
4.
5. // The terrains package contains the different terrains that are
6. // available within EvSim and the stats for that terrain, such as visibility
7. public class TerrainAttributes {
8.     public Random random = new Random(); // For random attributes such as temperature
9.     public String terrainId = "Default"; // The name of the terrain
10.    public double visibility = 1.0; // The visibility of the terrain
11.    public ArrayList<String> foods = new ArrayList<>(); // List of foods available within that
terrain
12.    public int temperature; // Temperature of the terrain
13. }
```

C3.1.2 Beach.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Beach extends TerrainAttributes {
4.     public Beach(){
5.         terrainId = "Beach";
6.         visibility = 1.1;
7.         foods.add("Beach Grass Seeds");
8.         foods.add("Shellfish and Crustaceans");
9.         foods.add("Coastal Fruit / Plants");
10.        temperature = random.nextInt(13, 23);
11.    }
12. }
13. }
```

C3.1.3 DeepWater.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class DeepWater extends TerrainAttributes {
4.     public DeepWater(){
5.         terrainId = "Deep Water";
6.         visibility = 0.4;
7.         foods.add("Plankton");
8.         foods.add("Deep Sea Fish");
9.         foods.add("Large Deep Sea Fish");
10.        temperature = random.nextInt(0, 4);
11.    }
12. }
```

C3.1.4 Forest.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Forest extends TerrainAttributes {
4.     public Forest() {
5.         terrainId = "Forest";
6.         visibility = 0.55;
7.         foods.add("Tree Spawn");
8.         foods.add("Berries");
9.         foods.add("Fungi");
10.        temperature = random.nextInt(20, 25);
11.    }
12. }
```

C3.1.5 Grass.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Grass extends TerrainAttributes {
4.     public Grass(){
5.         terrainId = "Grass";
6.         visibility = 1.0;
7.         foods.add("Grass Seeds");
8.         foods.add("Leaves / Large Plants");
9.         foods.add("Tree Roots / Plant Remains");
10.        temperature = random.nextInt(16, 22);
11.    }
12. }
```

C3.1.6 Mountain.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Mountain extends TerrainAttributes {
4.     public Mountain() {
5.         terrainId = "Mountain";
6.         visibility = 1.2;
7.         foods.add("Grasses and Herbs");
8.         foods.add("Shrubs");
9.         foods.add("Coniferous Tree Spawn");
10.        temperature = random.nextInt(4, 10);
11.    }
12. }
```

C3.1.7 Snow.java

```
1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Snow extends TerrainAttributes {
4.     public Snow() {
5.         terrainId = "Snow";
6.         visibility = 0.7;
```

```

7.         foods.add("Moss / Lichen");
8.         foods.add("Algae");
9.         foods.add("Shrub Berries");
10.        temperature = random.nextInt(-4, 4);
11.    }
12. }

```

C3.1.8 Water.java

```

1. package Terrains;
2. // Refer to TerrainAttributes.java for comments (C3.1.1 documentation)
3. public class Water extends TerrainAttributes {
4.     public Water(){
5.         terrainId = "Water";
6.         visibility = 0.6;
7.         foods.add("Aquatic Plants");
8.         foods.add("Algae");
9.         foods.add("Insects and Insect Larvae");
10.        temperature = random.nextInt(6, 13);
11.    }
12. }

```

C3.2 FoodTypes

For the food types section, the individual food classes are uncommented due to C3.2.1 being the parent, therefore containing comments that cover all other files. For more information on this package, see 3.3.

C3.2.1 FoodTypes.java

```

1. package FoodTypes;
2. // The parent FoodTypes class, which has 20 inheritors (all other food types)
3. public class FoodTypes {
4.     public String foodName; // The name of the food
5.     public double foodAvailability; // The availability
6.     public double foodFactor; // The factor
7.     public boolean waterBasedFood = false; // Whether it is land or food based
8.     public boolean landBasedFood = false;
9.     public String parentName; // The name of the parent food class (see 3.3.2 documentation)
10.    public String terrainName; // The name of the terrain where this food is found
11. }

```

C3.2.2 Algae.java

```

1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class Algae extends FoodTypes {
4.     public Algae(){
5.         parentName = "Algae and Aquatic Plants";
6.         foodName = "Algae";
7.         foodAvailability = 0.4;
8.         foodFactor = 0.25;
9.         terrainName = "Water";
10.        landBasedFood = false;
11.        waterBasedFood = true;
12.    }
13. }

```

C3.2.3 AquaticPlants.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class AquaticPlants extends FoodTypes {
4.     public AquaticPlants(){
5.         parentName = "Algae and Aquatic Plants";
6.         foodName = "Aquatic Plants";
7.         foodAvailability = 0.5;
8.         foodFactor = 0.2;
9.         terrainName = "Water";
10.        landBasedFood = false;
11.        waterBasedFood = true;
12.    }
13. }
```

C3.2.4 BeachGrassSeeds.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class BeachGrassSeeds extends FoodTypes {
4.     public BeachGrassSeeds(){
5.         parentName = "Plant Spawn";
6.         foodName = "Beach Grass Seeds";
7.         foodAvailability = 0.4;
8.         foodFactor = 0.25;
9.         terrainName = "Beach";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.5 Berries.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class Berries extends FoodTypes {
4.     public Berries(){
5.         parentName = "Fruits and Berries";
6.         foodName = "Berries";
7.         foodAvailability = 0.5;
8.         foodFactor = 0.15;
9.         terrainName = "Forest";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.6 CoastalFruitPlants.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class CoastalFruitPlants extends FoodTypes {
4.     public CoastalFruitPlants(){
5.         parentName = "Fruits and Berries";
6.         foodName = "Coastal Fruit / Plants";
7.         foodAvailability = 0.3;
8.         foodFactor = 0.4;
9.         terrainName = "Beach";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.7 ConiferousTreeSpawn.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class ConiferousTreeSpawn extends FoodTypes {
4.     public ConiferousTreeSpawn(){
5.         parentName = "Shrubs and Trees";
6.         foodName = "Coniferous Tree Spawn";
7.         foodAvailability = 0.3;
8.         foodFactor = 0.4;
```

```
9.         terrainName = "Mountain";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.8 DeepSeaFish.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class DeepSeaFish extends FoodTypes {
4.     public DeepSeaFish(){
5.         parentName = "Fish";
6.         foodName = "Deep Sea Fish";
7.         foodAvailability = 0.1;
8.         foodFactor = 0.5;
9.         terrainName = "Deep Water";
10.        landBasedFood = false;
11.        waterBasedFood = true;
12.    }
13. }
```

C3.2.9 Fungi.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class Fungi extends FoodTypes {
4.     public Fungi(){
5.         parentName = "Plant Spawn";
6.         foodName = "Fungi";
7.         terrainName = "Forest";
8.         foodAvailability = 0.3;
9.         foodFactor = 0.3;
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.10 GrassesHerbs.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class GrassesHerbs extends FoodTypes {
4.     public GrassesHerbs(){
5.         parentName = "Plant Spawn";
6.         terrainName = "Mountain";
7.         foodName = "Grasses and Herbs";
8.         foodAvailability = 0.4;
9.         foodFactor = 0.25;
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.11 GrassSeeds.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class GrassSeeds extends FoodTypes {
4.     public GrassSeeds(){
5.         parentName = "Plant Spawn";
6.         foodName = "Grass Seeds";
7.         foodAvailability = 0.6;
8.         foodFactor = 0.3;
9.         terrainName = "Grass";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.12 InsectsLarvae.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class InsectsLarvae extends FoodTypes {
```

```

4.     public InsectsLarvae(){
5.         parentName = "Aquatic life / Spawn";
6.         foodName = "Insects and Insect Larvae";
7.         foodAvailability = 0.2;
8.         foodFactor = 0.5;
9.         terrainName = "Water";
10.        landBasedFood = false;
11.        waterBasedFood = true;
12.    }
13.}

```

C3.2.13 LargeDeepSeaFish.java

```

1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class LargeDeepSeaFish extends FoodTypes {
4.     public LargeDeepSeaFish(){
5.         parentName = "Fish";
6.         foodName = "Large Deep Sea Fish";
7.         foodAvailability = 0.05;
8.         foodFactor = 0.8;
9.         terrainName = "Deep Water";
10.        landBasedFood = false;
11.        waterBasedFood = true;
12.    }
13.}

```

C3.2.14 LeavesPlants.java

```

1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class LeavesPlants extends FoodTypes {
4.     public LeavesPlants(){
5.         parentName = "Plant Parts";
6.         foodName = "Leaves / Large Plants";
7.         foodAvailability = 0.25;
8.         foodFactor = 0.4;
9.         terrainName = "Grass";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13.}

```

C3.2.15 MossLichen.java

```

1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class MossLichen extends FoodTypes {
4.     public MossLichen(){
5.         parentName = "Shrubs and Trees";
6.         foodName = "Moss / Lichen";
7.         foodAvailability = 0.4;
8.         foodFactor = 0.2;
9.         terrainName = "Snow";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13.}

```

C3.2.16 Plankton.java

```

1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class Plankton extends FoodTypes {
4.     public Plankton(){
5.         foodName = "Plankton";
6.         foodAvailability = 0.3;
7.         foodFactor = 0.1;
8.         landBasedFood = false;
9.         waterBasedFood = true;
10.        parentName = "Aquatic life / Spawn";
11.        terrainName = "Deep Water";
12.    }
13.}

```

C3.2.17 RootsPlantRemains.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class RootsPlantRemains extends FoodTypes {
4.     public RootsPlantRemains(){
5.         foodName = "Tree Roots / Plant Remains";
6.         foodAvailability = 0.15;
7.         foodFactor = 0.5;
8.         landBasedFood = true;
9.         waterBasedFood = false;
10.        parentName = "Plant Parts";
11.        terrainName = "Grass";
12.    }
13. }
```

C3.2.18 ShellfishCrustaceans.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class ShellfishCrustaceans extends FoodTypes {
4.     public ShellfishCrustaceans(){
5.         foodName = "Shellfish and Crustaceans";
6.         foodAvailability = 0.3;
7.         foodFactor = 0.35;
8.         landBasedFood = false;
9.         waterBasedFood = true;
10.        terrainName = "Beach";
11.        parentName = "Aquatic Life / Spawn";
12.    }
13. }
```

C3.2.19 ShrubBerries.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class ShrubBerries extends FoodTypes {
4.     public ShrubBerries(){
5.         foodName = "Shrub Berries";
6.         foodAvailability = 0.3;
7.         foodFactor = 0.45;
8.         landBasedFood = true;
9.         waterBasedFood = false;
10.        terrainName = "Snow";
11.        parentName = "Fruits and Berries";
12.    }
13. }
```

C3.2.20 Shrubs.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class Shrubs extends FoodTypes {
4.     public Shrubs(){
5.         parentName = "Shrubs and Trees";
6.         foodName = "Shrubs";
7.         foodAvailability = 0.3;
8.         foodFactor = 0.35;
9.         terrainName = "Mountain";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }
```

C3.2.21 TreeSpawn.java

```
1. package FoodTypes;
2. // Refer to FoodTypes.java for comments (C3.2.1 documentation)
3. public class TreeSpawn extends FoodTypes {
4.     public TreeSpawn(){
5.         parentName = "Shrubs and Trees";
6.         foodName = "Tree Spawn";
7.         foodAvailability = 0.4;
8.         foodFactor = 0.25;
```

```

9.         terrainName = "Forest";
10.        landBasedFood = true;
11.        waterBasedFood = false;
12.    }
13. }

```

C4 Storage

This section covers the backend to EvSim, more specifically the JSON files storing both user options and organisms. These two sections are variable throughout the program, due to them being a backend store, the blocks provided here are uncommented (data store) and are simply examples.

C4.1 options.json

```

1. {
2.     "colours": [
3.         {
4.             "red": 0,
5.             "green": 153,
6.             "blue": 0,
7.             "setting": "Trichromacy (2)"
8.         },
9.         {
10.            "red": 0,
11.            "green": 128,
12.            "blue": 128,
13.            "setting": "Deutanopia (1)"
14.        },
15.        {
16.            "red": 160,
17.            "green": 160,
18.            "blue": 160,
19.            "setting": "Deutanopia (2)"
20.        },
21.        {
22.            "red": 128,
23.            "green": 128,
24.            "blue": 0,
25.            "setting": "Deutanopia (1)"
26.        },
27.        {
28.            "red": 255,

```

```
29.          "green": 255,
30.          "blue": 255,
31.          "setting": "Monochromacy (2)"
32.      },
33.      {
34.          "red": 205,
35.          "green": 180,
36.          "blue": 255,
37.          "setting": "Tritanopia (1)"
38.      },
39.      {
40.          "red": 125,
41.          "green": 14,
42.          "blue": 7,
43.          "setting": null
44.      }
45. ],
46. "windowWidth": 1920,
47. "windowHeight": 1080
48. }
```

C4.2 organisms.json

```
1. [
2.     {
3.         "organismId": 0,
4.         "stats": {
5.             "currentAwareness": 2.0,
6.             "organismsEncountered": 0.5,
7.             "awarenessHistory": [
8.                 1.5,
9.                 1.5,
10.                1.5,
11.                1.5
12.            ],
13.            "age": 57,
```

```

14.         "preferredFood": {
15.             "foodName": "Shellfish and Crustaceans",
16.             "foodAvailability": 0.3,
17.             "foodFactor": 0.35,
18.             "waterBasedFood": true,
19.             "landBasedFood": false,
20.             "parentName": "Aquatic Life / Spawn",
21.             "terrainName": "Beach"
22.         },
23.         "preferredFoodString": null,
24.         "eatenFood": {
25.             "foodName": "Tree Roots / Plant Remains",
26.             "foodAvailability": 0.15,
27.             "foodFactor": 0.5,
28.             "waterBasedFood": false,
29.             "landBasedFood": true,
30.             "parentName": "Plant Parts",
31.             "terrainName": "Grass"
32.         },
33.         "recentFood": [],
34.         "foodFactor": 0.15,
35.         "sizeFactor": 25.30000000000022,
36.         "size": 0.5,
37.         "speed": 12.33,
38.         "startSpeed": 5.0,
39.         "currentTemp": 7,
40.         "preferredTemp": 22,
41.         "recentTemp": [],
42.         "tempFactor": 0.4,
43.         "preferredTerrain": "Beach",
44.         "secondPreferredTerrain": "Grass",
45.         "thirdPreferredTerrain": "Default",
46.         "health": 10.2,
47.         "dead": false,
48.         "ageLastBred": 15,
49.         "generation": 1,
50.         "organismsMet": 3,
51.         "totalOrganismsMet": 10,
52.         "tsb": null,
53.         "lastAttack": 10
54.     },
55.     "currentTerrain": "Beach",
56.     "xCoordinate": 817,
57.     "yCoordinate": 530,
58.     "sex": 0,
59.     "fatherID": -1,
60.     "motherID": -1,
61.     "predator": false
62. },
63. {
64.     "organismId": 1,
65.     "stats": {
66.         "currentAwareness": 1.91,
67.         "organismsEncountered": 0.5,
68.         "awarenessHistory": [
69.             1.6,
70.             1.6,
71.             1.5,
72.             1.4
73.         ],
74.         "age": 57,
75.         "preferredFood": {
76.             "foodName": "Grass Seeds",
77.             "foodAvailability": 0.6,
78.             "foodFactor": 0.3,
79.             "waterBasedFood": false,
80.             "landBasedFood": true,
81.             "parentName": "Plant Spawn",
82.             "terrainName": "Grass"
83.         },
84.         "preferredFoodString": null,

```

```

85.         "eatenFood": {
86.             "foodName": "Beach Grass Seeds",
87.             "foodAvailability": 0.4,
88.             "foodFactor": 0.25,
89.             "waterBasedFood": false,
90.             "landBasedFood": true,
91.             "parentName": "Plant Spawn",
92.             "terrainName": "Beach"
93.         },
94.         "recentFood": [],
95.         "foodFactor": 0.2,
96.         "sizeFactor": 16.05000000000001,
97.         "size": 0.5,
98.         "speed": 12.12,
99.         "startSpeed": 5.0,
100.        "currentTemp": 8,
101.        "preferredTemp": 19,
102.        "recentTemp": [],
103.        "tempFactor": 0.56,
104.        "preferredTerrain": "Grass",
105.        "secondPreferredTerrain": "Beach",
106.        "thirdPreferredTerrain": "Forest",
107.        "health": 12.41,
108.        "dead": false,
109.        "ageLastBred": 55,
110.        "generation": 1,
111.        "organismsMet": 1,
112.        "totalOrganismsMet": 7,
113.        "tsb": null,
114.        "lastAttack": 10
115.    },
116.    "currentTerrain": "Beach",
117.    "xCoordinate": 804,
118.    "yCoordinate": 554,
119.    "sex": 0,
120.    "fatherID": -1,
121.    "motherID": -1,
122.    "predator": false
123. },
124. {
125.     "organismId": 2,
126.     "stats": {
127.         "currentAwareness": 2.1,
128.         "organismsEncountered": 0.5,
129.         "awarenessHistory": [
130.             1.6,
131.             1.6,
132.             1.6,
133.             1.6
134.         ],
135.         "age": 57,
136.         "preferredFood": {
137.             "foodName": "Aquatic Plants",
138.             "foodAvailability": 0.5,
139.             "foodFactor": 0.2,
140.             "waterBasedFood": true,
141.             "landBasedFood": false,
142.             "parentName": "Algae and Aquatic Plants",
143.             "terrainName": "Water"
144.         },
145.         "preferredFoodString": null,
146.         "eatenFood": {
147.             "foodName": "Coastal Fruit / Plants",
148.             "foodAvailability": 0.3,
149.             "foodFactor": 0.4,
150.             "waterBasedFood": false,
151.             "landBasedFood": true,
152.             "parentName": "Fruits and Berries",
153.             "terrainName": "Beach"
154.         },
155.         "recentFood": []

```

```

156.         "foodFactor": 0.12,
157.         "sizeFactor": 16.550000000000004,
158.         "size": 0.5,
159.         "speed": 11.39,
160.         "startSpeed": 5.0,
161.         "currentTemp": 14,
162.         "preferredTemp": 13,
163.         "recentTemp": [],
164.         "tempFactor": 0.96,
165.         "preferredTerrain": "Water",
166.         "secondPreferredTerrain": "Beach",
167.         "thirdPreferredTerrain": "Default",
168.         "health": 11.37,
169.         "dead": false,
170.         "ageLastBred": 55,
171.         "generation": 1,
172.         "organismsMet": 5,
173.         "totalOrganismsMet": 57,
174.         "tsb": null,
175.         "lastAttack": 10
176.     },
177.     "currentTerrain": "Beach",
178.     "xCoordinate": 800,
179.     "yCoordinate": 559,
180.     "sex": 1,
181.     "fatherID": -1,
182.     "motherID": -1,
183.     "predator": false
184.   },
185.   {
186.     "organismId": 3,
187.     "stats": {
188.       "currentAwareness": 1.6,
189.       "organismsEncountered": 0.5,
190.       "awarenessHistory": [
191.         1.1,
192.         1.1,
193.         1.1,
194.         1.1
195.       ],
196.       "age": 57,
197.       "preferredFood": {
198.         "foodName": "Aquatic Plants",
199.         "foodAvailability": 0.5,
200.         "foodFactor": 0.2,
201.         "waterBasedFood": true,
202.         "landBasedFood": false,
203.         "parentName": "Algae and Aquatic Plants",
204.         "terrainName": "Water"
205.       },
206.       "preferredFoodString": null,
207.       "eatenFood": {
208.         "foodName": "Aquatic Plants",
209.         "foodAvailability": 0.5,
210.         "foodFactor": 0.2,
211.         "waterBasedFood": true,
212.         "landBasedFood": false,
213.         "parentName": "Algae and Aquatic Plants",
214.         "terrainName": "Water"
215.       },
216.       "recentFood": [],
217.       "foodFactor": 0.2,
218.       "sizeFactor": 11.49999999999993,
219.       "size": 0.5,
220.       "speed": 11.23,
221.       "startSpeed": 5.0,
222.       "currentTemp": -3,
223.       "preferredTemp": 10,
224.       "recentTemp": [],
225.       "tempFactor": 0.48,
226.       "preferredTerrain": "Water",

```

```

227.         "secondPreferredTerrain": "Default",
228.         "thirdPreferredTerrain": "Default",
229.         "health": 10.84,
230.         "dead": false,
231.         "ageLastBred": 33,
232.         "generation": 1,
233.         "organismsMet": 3,
234.         "totalOrganismsMet": 40,
235.         "tsb": null,
236.         "lastAttack": 10
237.     },
238.     "currentTerrain": "Water",
239.     "xCoordinate": 751,
240.     "yCoordinate": 539,
241.     "sex": 0,
242.     "fatherID": -1,
243.     "motherID": -1,
244.     "predator": false
245.   },
246.   {
247.     "organismId": 4,
248.     "stats": {
249.       "currentAwareness": 2.1,
250.       "organismsEncountered": 0.5,
251.       "awarenessHistory": [
252.         1.6,
253.         1.6,
254.         1.6,
255.         1.6
256.       ],
257.       "age": 57,
258.       "preferredFood": {
259.         "foodName": "Shellfish and Crustaceans",
260.         "foodAvailability": 0.3,
261.         "foodFactor": 0.35,
262.         "waterBasedFood": true,
263.         "landBasedFood": false,
264.         "parentName": "Aquatic Life / Spawn",
265.         "terrainName": "Beach"
266.       },
267.       "preferredFoodString": null,
268.       "eatenFood": {
269.         "foodName": "Shellfish and Crustaceans",
270.         "foodAvailability": 0.3,
271.         "foodFactor": 0.35,
272.         "waterBasedFood": true,
273.         "landBasedFood": false,
274.         "parentName": "Aquatic Life / Spawn",
275.         "terrainName": "Beach"
276.       },
277.       "recentFood": [],
278.       "foodFactor": 0.35,
279.       "sizeFactor": 24.65000000000013,
280.       "size": 0.5,
281.       "speed": 12.45,
282.       "startSpeed": 5.0,
283.       "currentTemp": 8,
284.       "preferredTemp": 23,
285.       "recentTemp": [],
286.       "tempFactor": 0.4,
287.       "preferredTerrain": "Beach",
288.       "secondPreferredTerrain": "Grass",
289.       "thirdPreferredTerrain": "Water",
290.       "health": 8.33,
291.       "dead": false,
292.       "ageLastBred": 43,
293.       "generation": 1,
294.       "organismsMet": 4,
295.       "totalOrganismsMet": 74,
296.       "tsb": null,
297.       "lastAttack": 10

```

```

298.        },
299.        "currentTerrain": "Beach",
300.        "xCoordinate": 811,
301.        "yCoordinate": 520,
302.        "sex": 1,
303.        "fatherID": -1,
304.        "motherID": -1,
305.        "predator": false
306.    },
307.    {
308.        "organismId": 5,
309.        "stats": {
310.            "currentAwareness": 1.6,
311.            "organismsEncountered": 0.5,
312.            "awarenessHistory": [
313.                1.1,
314.                1.1,
315.                1.1,
316.                1.1
317.            ],
318.            "age": 29,
319.            "preferredFood": {
320.                "foodName": "Aquatic Plants",
321.                "foodAvailability": 0.5,
322.                "foodFactor": 0.2,
323.                "waterBasedFood": true,
324.                "landBasedFood": false,
325.                "parentName": "Algae and Aquatic Plants",
326.                "terrainName": "Water"
327.            },
328.            "preferredFoodString": null,
329.            "eatenFood": {
330.                "foodName": "Aquatic Plants",
331.                "foodAvailability": 0.5,
332.                "foodFactor": 0.2,
333.                "waterBasedFood": true,
334.                "landBasedFood": false,
335.                "parentName": "Algae and Aquatic Plants",
336.                "terrainName": "Water"
337.            },
338.            "recentFood": [],
339.            "foodFactor": 0.2,
340.            "sizeFactor": 5.800000000000025,
341.            "size": 0.5,
342.            "speed": 7.35,
343.            "startSpeed": 4.5,
344.            "currentTemp": 1,
345.            "preferredTemp": 19,
346.            "recentTemp": [],
347.            "tempFactor": 0.28,
348.            "preferredTerrain": "Water",
349.            "secondPreferredTerrain": "Default",
350.            "thirdPreferredTerrain": "Default",
351.            "health": 50.74,
352.            "dead": false,
353.            "ageLastBred": 43,
354.            "generation": 2,
355.            "organismsMet": 0,
356.            "totalOrganismsMet": 56,
357.            "tsb": null,
358.            "lastAttack": -1
359.        },
360.        "currentTerrain": "Water",
361.        "xCoordinate": 812,
362.        "yCoordinate": 503,
363.        "sex": 0,
364.        "fatherID": 2,
365.        "motherID": 3,
366.        "predator": false
367.    },
368.    {

```

```

369.     "organismId": 6,
370.     "stats": {
371.         "currentAwareness": 1.6,
372.         "organismsEncountered": 0.5,
373.         "awarenessHistory": [
374.             1.1,
375.             1.1,
376.             1.1,
377.             1.1
378.         ],
379.         "age": 23,
380.         "preferredFood": {
381.             "foodName": "Aquatic Plants",
382.             "foodAvailability": 0.5,
383.             "foodFactor": 0.2,
384.             "waterBasedFood": true,
385.             "landBasedFood": false,
386.             "parentName": "Algae and Aquatic Plants",
387.             "terrainName": "Water"
388.         },
389.         "preferredFoodString": null,
390.         "eatenFood": {
391.             "foodName": "Aquatic Plants",
392.             "foodAvailability": 0.5,
393.             "foodFactor": 0.2,
394.             "waterBasedFood": true,
395.             "landBasedFood": false,
396.             "parentName": "Algae and Aquatic Plants",
397.             "terrainName": "Water"
398.         },
399.         "recentFood": [],
400.         "foodFactor": 0.2,
401.         "sizeFactor": 4.600000000000001,
402.         "size": 0.5,
403.         "speed": 6.93,
404.         "startSpeed": 4.5,
405.         "currentTemp": -2,
406.         "preferredTemp": 5,
407.         "recentTemp": [],
408.         "tempFactor": 0.72,
409.         "preferredTerrain": "Water",
410.         "secondPreferredTerrain": "Default",
411.         "thirdPreferredTerrain": "Default",
412.         "health": 64.58,
413.         "dead": false,
414.         "ageLastBred": -1,
415.         "generation": 2,
416.         "organismsMet": 0,
417.         "totalOrganismsMet": 23,
418.         "tsb": null,
419.         "lastAttack": -1
420.     },
421.     "currentTerrain": "Water",
422.     "xCoordinate": 760,
423.     "yCoordinate": 508,
424.     "sex": 1,
425.     "fatherID": 4,
426.     "motherID": 3,
427.     "predator": false
428. },
429. {
430.     "organismId": 7,
431.     "stats": {
432.         "currentAwareness": 1.6,
433.         "organismsEncountered": 0.5,
434.         "awarenessHistory": [
435.             1.1,
436.             1.1,
437.             1.1,
438.             1.1
439.         ],

```

```

440.         "age": 13,
441.         "preferredFood": {
442.             "foodName": "Shellfish and Crustaceans",
443.             "foodAvailability": 0.3,
444.             "foodFactor": 0.35,
445.             "waterBasedFood": true,
446.             "landBasedFood": false,
447.             "parentName": "Aquatic Life / Spawn",
448.             "terrainName": "Beach"
449.         },
450.         "preferredFoodString": null,
451.         "eatenFood": {
452.             "foodName": "Insects and Insect Larvae",
453.             "foodAvailability": 0.2,
454.             "foodFactor": 0.5,
455.             "waterBasedFood": true,
456.             "landBasedFood": false,
457.             "parentName": "Aquatic life / Spawn",
458.             "terrainName": "Water"
459.         },
460.         "recentFood": [],
461.         "foodFactor": 0.3,
462.         "sizeFactor": 6.5,
463.         "size": 0.5,
464.         "speed": 6.85,
465.         "startSpeed": 4.5,
466.         "currentTemp": -3,
467.         "preferredTemp": 17,
468.         "recentTemp": [],
469.         "tempFactor": 0.1999999999999996,
470.         "preferredTerrain": "Beach",
471.         "secondPreferredTerrain": "Water",
472.         "thirdPreferredTerrain": "Default",
473.         "health": 80.28,
474.         "dead": false,
475.         "ageLastBred": -1,
476.         "generation": 2,
477.         "organismsMet": 0,
478.         "totalOrganismsMet": 15,
479.         "tsb": null,
480.         "lastAttack": -1
481.     },
482.     "currentTerrain": "Water",
483.     "xCoordinate": 807,
484.     "yCoordinate": 492,
485.     "sex": 1,
486.     "fatherID": 4,
487.     "motherID": 5,
488.     "predator": false
489. },
490. {
491.     "organismId": 8,
492.     "stats": {
493.         "currentAwareness": 1.5,
494.         "organismsEncountered": 0.2,
495.         "awarenessHistory": [
496.             1.3
497.         ],
498.         "age": 1,
499.         "preferredFood": {
500.             "foodName": "Aquatic Plants",
501.             "foodAvailability": 0.5,
502.             "foodFactor": 0.2,
503.             "waterBasedFood": true,
504.             "landBasedFood": false,
505.             "parentName": "Algae and Aquatic Plants",
506.             "terrainName": "Water"
507.         },
508.         "preferredFoodString": null,
509.         "eatenFood": {
510.             "foodName": "Coastal Fruit / Plants",

```

```
511.         "foodAvailability": 0.3,
512.         "foodFactor": 0.4,
513.         "waterBasedFood": false,
514.         "landBasedFood": true,
515.         "parentName": "Fruits and Berries",
516.         "terrainName": "Beach"
517.     },
518.     "recentFood": [],
519.     "foodFactor": 0.12,
520.     "sizeFactor": 0.4,
521.     "size": 0.6,
522.     "speed": 5.55,
523.     "startSpeed": 5.5,
524.     "currentTemp": 0,
525.     "preferredTemp": 11,
526.     "recentTemp": [
527.         13
528.     ],
529.     "tempFactor": 1.0,
530.     "preferredTerrain": "Water",
531.     "secondPreferredTerrain": "Default",
532.     "thirdPreferredTerrain": "Default",
533.     "health": 98.62,
534.     "dead": false,
535.     "ageLastBred": -1,
536.     "generation": 2,
537.     "organismsMet": 0,
538.     "totalOrganismsMet": 2,
539.     "tsb": null,
540.     "lastAttack": -1
541.   },
542.   "currentTerrain": "Beach",
543.   "xCoordinate": 799,
544.   "yCoordinate": 562,
545.   "sex": 1,
546.   "fatherID": 2,
547.   "motherID": 1,
548.   "predator": false
549. }
550. ]
```