Design page:

Breaking down the solution into different functions

User interface design

Initial idea:

A picture containing graphical user interface

Description automatically generated

Initial idea 1 (start page):

A picture containing text

Description automatically generated

Initial idea 2 (option page):

Graphical user interface, diagram

Description automatically generated with medium confidence

Developed sample 1 (start page):

Chart, waterfall chart

Description automatically generated

Developed sample 2(start page):

Graphical user interface

Description automatically generated with medium confidence

Developed sample 3 (with top bar disabled):

Graphical user interface

Description automatically generated

This is the main simulation page of the program, it allows the user to visualise what’s happening and also see more details about what’s happening in the simulation log. The start page allows for the simulation to be started or edited before hand.

**User interface customisation and accessibility page:**

Text

Description automatically generated

Graphical user interface

Description automatically generated

**Explanation:**

This page controls the accessibility settings for the program whilst also giving users the option to change any part of the UI’s theme to suit their preferences

Each colour element Is customisable using the inbuilt colour editor which allows the choice of any colour and a preview of the colour before being selected, the color selector has a button to flip the colour scale in order to display black and white as an option.

**Start page:**

Start button:

This will bring the user to the simulation page and start the simulation at the same time

Back button:

This allows for the user to the return the start page at any time to edit anything, and then resume the simulation at the exact same state

Theme button:

This contains the accessibility settings for colour blindness and also an ability to change the font to larger or smaller to allow for some users to be able to read it. These can also be used by any user to change the colour of UI elements to their preference

Quit button:

A replacement for the inbuilt exit button within windows, since the new design removes the top bar it is required to quit the program, on exit the program saves the users preference, the state of the simulation and log in a file

**Simulation log:**

Outline drop down:

Allows the user to highlight certain details they are looking for such as when a creature reproduces, it also prevents the log entry from being timed out to prevent the user losing import data although it is to be noted the log is saved to a file so no data is truly lost.

Filter drop down:

Allows the user to prevent information on the log from appearing if they do not want It to appear for instance if they only wanted to view mutations then they could filter all elements except mutations.

**environment control options:**

environment sliders:

allows the user to control the environment in accordance to their needs using slider elements that are simple and adhere to their use by taking in a maximum and minimum value aswell as a maximum value

environment bars:

shows the metrics of the simulation whilst its running in a simple overview manner, I found in my stakeholder feedback that a graph being drawn as the simulation is run would be too confusing to some users and should be reserved for viewing after the simulation has finished

Ui element flowchart:

Chart, box and whisker chart

Description automatically generated

**User interface of other solutions**

Chart, bubble chart

Description automatically generated

**what have they done?**

This solution utilises a key on the bottom of the user interface, to explain what color represents which organism. The solution also has a line of text explaining the controls of the simulation, both the key and the text are laid out on a darker coloured rectangle outside of the simulation area to clearly highlight its presence. This solution uses the default windows border for displaying the window allowing minimisation, maximisation and adjustment.

**How does this differ to my solution?**

My solution doesn’t use the windows border for adjustment, instead the gui is frameless and adjustable via appropriately coloured customised buttons. My solution also currently does not have a key or text explaining the controls and instead of using a different colour to highlight different areas they are bordered and appropriately labelled.

**What can I learn from this solution?**

Investigating this solution has allowed me to adjust the gui of my solution with it now including a key explaining the different organisms and a toggleable area that displays a paragraph explaining how to control the simulation.



**what have they done?**

This solution features multiple buttons allowing for manipulation of the environment including for time manipulation, heat, light, access to diagrams. The solution maintains the same blue on black colouring throughout with slider elements, buttons and overlay panels for specific manipulation of organisms

**how does this differ from my solution?**

the solution differs from my solution due to

**algorithms:**

|  |  |  |
| --- | --- | --- |
| **input** | **Process** | **output** |
| Home page buttons excluding simulation start | Trigger option tab, which will then display all relevant UI elements | the corresponding variables are updated from the input of the user |
| Organism objects | Iterate through objects and simulate natural selection by manipulating organisms | Simulation is displayed on the screen, the simulation data is also updated on the graphs |
| User input on the simulation page | Save settings to file, update corresponding variables | the simulation environment is updated accordingly |
| Exit key press | Pygame exit function called, code is stopped and json file is saved | The pygame window closes and the user inputs are saved |
| Genetic information of organisms | Species are sorted into a tree corresponding to common ancestors and direct descendants | Species Diagram tree drawn on the screen |
| Log entries | the log entries are iterated through, drop down text is checked for mouse hovering | log entries displayed on screen with yellow text for extra information |

**key algorithm 1:**

the main simulation loop

flowchart:

Diagram

Description automatically generated

The subroutines to this process are as follows:

* Create\_move
* Sight\_check
* Reproduce
* Wander
* Target check

Breakdown of individual subprocesses

**Create\_move**

Inputs: no inputs into the function since its handled inside the herbivore class and all appropriate arguments are stored inside

Return value: none, its purpose is to update the position of the creature to its target and so no value is needed to be returned

Pseudocode:

FUNCTION create\_move(this)

Try

Food\_object\_array[food\_object\_array.index(this.target)] // tries to locate the // target in the array to make sure its there

Except

Del this.target // if the target isn’t there then its deleted to prevent an error

return

this.pos = tangent(this.pos,this.target.pos,this.rotation,1) // update the position

this.rotation = rad\_to\_deg(this.pos,this.target.pos) // update the rotation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| This.pos | This.target.pos | Target in array | This.rotation | Expected output | output |
| [100,100] | NULL | False | 0 | Return | Return |
| [-240,0] | NULL | False | 30 | return | Return |
| [500,200] | [100,100] | True | 60 | continued | Continued |
| [205,40] | [203,243] | True | -30 | Returned ( tangent function) | continued |

**Why this test data?**

I used this test data to prevent the program from crashing when given invalid arguments, when a an objects position is perfectly adjacent/horizontal the rad\_to\_deg will return a negative angle due it the function not being able to make a perfect triangle, I used this data to prevent complications with this scenario.

What can I learn from this output:

The rad\_to\_deg function still accepts invalid rotations, a check has been implemented to fix this with

other functions used:

tangent : returns the position along a tangent 1 away from the host aligned with the nose of the creature.

rad\_to\_deg : returns the relative angle from two position vectors

explanation:

the function is used to move the creature one step closer to the target e.g food or another creature

**sight\_check**

inputs: the target vec to be checked, the rest are contained within the creature’s class

return value: returns a Boolean value as to whether the creature is in the vision cone

pseudocode:

FUNCTION sight\_check(this,target)

Point\_1 = tangent(this.pos,this.rotation+270,50)

Point\_2 = tangent(this.pos,this.rotation+90,50)

Point\_3 = tangent(this.pos,this.rotation,50)

Return is\_in\_triangle(Point\_1,Point\_2,Point\_3,target)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| target | Point\_1 | Point\_2 | Point\_3 | Expected output | output |
| 650,227 | [511, 319] | [694, 318] | [603, 227] | True | True |
| [230, 266] | [190, 224] | [210, 283 | [230, 244] | True | True |
| [439, 431] | [407, 400] | [463, 455] | [408, 511] | False | False |
| [447, 472] | [407, 400] | [408, 511] | [463, 455] | False | False |

**Why this test data?**

I used this test data to test the functionality of the function, the data entered into the function corresponds to possible scenarios the function may be called in. by yielding an output for this test data I can test whether the function returns the right output in the right scenarios.

Other functions used:

* Is\_in\_triangle : returns whether a coordinate is inside a triangle using a formula

Explanation:

A triangle is made around the creature using a tangent at different rotations, the different points are then passed to a function to return whether the coordinates for the target are inside

Use inside simulation:

Is called to check whether an organism can seen another organism, is used by carnivores every tick to allow herbivores to run away without being chased and is called by herbivores to locate new food

**Reproduce:**

Inputs: no inputs except the host class

Return: no return value

Pseudo:

FUNCTION reproduce(this):

For I in range(this.litter)

Offspring = herbivore()

Offspring.genotype = this.genotype

Offspring.genotype = create\_mutation(offspring.genotype)

Entity\_object\_array.append(offspring)

Other functions used:

* Create\_mutation : used to simulate the alteration of DNA, in this case it is being used to create genetic variation

Explanation:

in this algorithm an offspring is made and the parents genotype is written to it, afterwards genetic variation caused by reproduction is simulated using a mutation and then the offspring are added to the register

Use in simulation

Reproduction is a key part of simulating natural selection as its essential to the idea of brute force adaptations and adaptations being passed generation to generation.

**Wander**

Inputs: none

Returns: none

Pseudo:

FUNCTION wander(this)

If (this.dummy = NULL)

If (this.stomach > this.stomach\_max\*0.8)

This.dummy = dummy(flags:HABITAT)

else

This.dummy = dummy(flags:RANDOM)

If (distance\_to(this.pos,this.dummy.pos) > 3)

This.dummy.release()

return

This.pos = tangent(this.pos,this.dummy.pos,1)

This.rotation = rad\_to\_deg(this.pos,this.dummy.pos)

**Test data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| dummy | pos | stomach | Stomach\_max | Expected result | result |
| NULL | (irrelevant) | 20 | 60 | Randomised dummy |  |
| NULL | (irrelevant) | 30 | 60 | Randomised dummy |  |
| NULL | (irrelevant) | 50 | 60 | Dummy on habitat |  |
| \*isn’t null\* pos: [100,200] | [100,100] | (irrelevant) | (irrelevant) | Continued movement |  |
| \*isn’t null\* pos: [200,200] | [205,200] | (irrelevant) | (irrelevant) | Continued movement |  |
| \*isn’t null\* pos: [300,300] | [300,300] | (irrelevant) | (irrelevant) | Dummy released |  |

**Why this test data?**

The function is intented to give 3 different outcomes depending on the conditions where the function is called, this test data is intended to test the system to preventing crashing by presenting an invalid target for the organism to move to therefore causing a crash if the issue is not caught. The test data is also intended to test whether the function recognises when to generate a new target and what type of target to generate whether it being randomised or in the species habitat.

Other functions used:

Tangent : used for movement forward in the current direction

Rad\_to\_deg : used to update the rotation every tick to smoothen the movement instead of straight lines

Explanation:

First, a dummy target is created at either random location to scout for food, or at the habitat for reproduction

Use in simulation:

Used to look for food when it currently isn’t moving towards one

KEY variables (global)

Entity\_object\_array:

* Type: array
* Use: stores all the herbivore obects in an array to be traversed by target selection algorithms

Hunter\_object\_array:

* Type: array
* Use: stores all the carnivore obects in an array to be traversed

Food\_object\_array:

* Type: array
* Use: stores all the food obects in an array to be traversed by target selection algorithms

Log\_index:

* Type: array
* Use: stores all the log entries which are stored as object type, by storing as object type each entry can have different attributes such as special text with a drop down menu

Lag\_comp:

* Type: float
* Use: stores the result of a math equation to calculate the multiplier of simulation elements, for instance if the simulation is running slowly then the hitboxes and speed of creatures should be increased to ensure the same speed at all times

Tab:

* Type: int
* Use: stores the current ui tab for switching between, prevents a potential issue where 2 pages are loaded at the same time since only one page can be represented by int

Bases:

* Type: array
* Uses: stores the dna bases in an array for random selection by mutation functions or generating a genome

Color variables e.g vgui\_entity\_herbivore :

* Type: color vector
* Use: stores all of the colors used to for the gui, allows for colors to be edited in themes for accessibility or personal preference

Ui variables :

* Type: objects
* Use: allows for efficient design as the ui elements can be reused without storing all of the appropriate states and variables in individual variables, also allows for all ui elements to be stored in an array and drawn using .draw() and then their states to be read directly elsewhere. This allows for code efficiency.

Class (private variables)

Organisms:

* **this.strand\_stomach**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it stores the information on how much capacity the organisms stomach can hold
* **this.strand\_bmr**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it dictates the metabolic activity of the organism e.g how fast it loses and gains energy stores
* **this.strand\_sight**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it decides how far and wide the organism can see
* **this.strand\_speed**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it decides how fast the organism can move
* **this.strand\_reprod**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it decides the time synthesize an egg
* **this.strand\_litter\_size**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it decides the amount of eggs it will lay at a time
* **this.strand\_total\_genotype**
  + Stores: a strand of raw DNA comprised of the bases stored in the array ‘bases’, in this case it stores the genotype acting as an identifier of the species
* **this.speed**
  + Stores the numerical value as an int of the organisms speed derived from the DNA strand
* **this.pos**
  + Stores the current 2d position of the organism on the display as a vector
* **this.litter\_size**
  + Stores the numerical value as an integer of the organisms litter size derived from the DNA strand
* **this.rotation**
  + Stores the current rotation of the nose and therefore the vision cone as a floating point
* **this.sight**
  + Stores the integer value of the organisms sight derived from the DNA strand
* **this.stomach**
  + Stores the current stomach calorie contents as an integer
* **this.stomach\_max**
  + Stores the derived result from the DNA strand of the organisms max stomach contents as an integer
* **this.bmr**
  + Stores the integer value of the organisms metabolic rate derived from the DNA strand
* **this.target**
  + Stores the current target stored as an organism class
* **this.wait\_for**
  + How long the organism should wait for after a decision stored as an int
* **this.dummy**
  + Stores a fake target for the organism to move to when it cant find any food as a dummy type
* **this.egg\_progress**
  + Stores the current progress of making an egg as a float
* **this.nutrition**
  + Stores the current body mass of the organism and its calorific contents as a float
* **this.dead**
  + Bool value of whether an organism is decomposing
* **this.nose**
  + The nose position of the organism so it can be read by elsewhere, its stored as a vector and currently is used to optimise the target selection algorithm

**Button:**

* **This.pos:**
  + Stores the position of the button as a vector, refers to the top right corner of the position
* **This.label:**
  + Stores a string displayed on the button showing its purpose

**Slider:**

* **This.pos:**
  + Stores the sliders position as a vector
* **This.label:**
  + Stores a string displayed under the slider showing what it does
* **This.min:**
  + Stores the minimum value of the slider as an integer
* **This.max:**
  + Stores the maximum value of the slider as an integer
* **This. Val:**
  + The translated slider position from the value returned, basically stores the position of the sliding element

**Checkbox:**

* **This.pos:**
  + Stores the position of the checkbox element as a vector
* **This.label**
  + Stores a string for displaying next to the checkbox saying what it changes
* **This.State**
  + The Boolean state of the checkbox
* **This.c\_state**
  + The last mouse state since activation to stop repeated activations when the mouse is held down, acts as a workaround since pygames GetMouseState doesn’t support bitflagging. Stored as a Boolean
* **This.hovered** 
  + Whether the cursor is above the checkbox element stored as a Boolean

**Color\_selector:**

* **this.pos:**
  + stored as a vector and stores the position of the color selector
* **this.col:**
  + stores the color currently selected from the reference variable stored as a 3d vector
* **this.state:**
  + whether the color element is opened to show the selection interface stored as a Boolean
* **this.c\_state:**
  + stores the last mousestate as a Boolean to prevent the activation button being repeatably pressed when the mouse is held down
* **this.invert:**
  + whether the color selector is set to grayscale to show a different color spectrum

**Selection\_interface\_s:**

* **this.pos:**
  + a vector storing the current UI elements position
* **this.selections:**
  + an array storing all possible selections
* **this.Selected**
  + a pointer to the selected item
* **this.state**
  + whether the interface is open, stored as a Boolean
* **this.c\_state:**
  + the last mouse state stored as a Boolean, to prevent multiple activations of the interface at once

**Selection\_interface\_m:**

* **this.pos:**
  + a vector storing the position of the interface
* **this.selections**
  + an array of all possible selections
* **this.selected**
  + an array of Boolean states so a selection can be referenced to check its state
* **this.state**
  + whether the interface is open, stored as a Boolean
* **this.c\_state:**
  + the last mouse state stored as a Boolean, to prevent multiple activations of the interface at once

**class diagram:**

**Diagram

Description automatically generated**