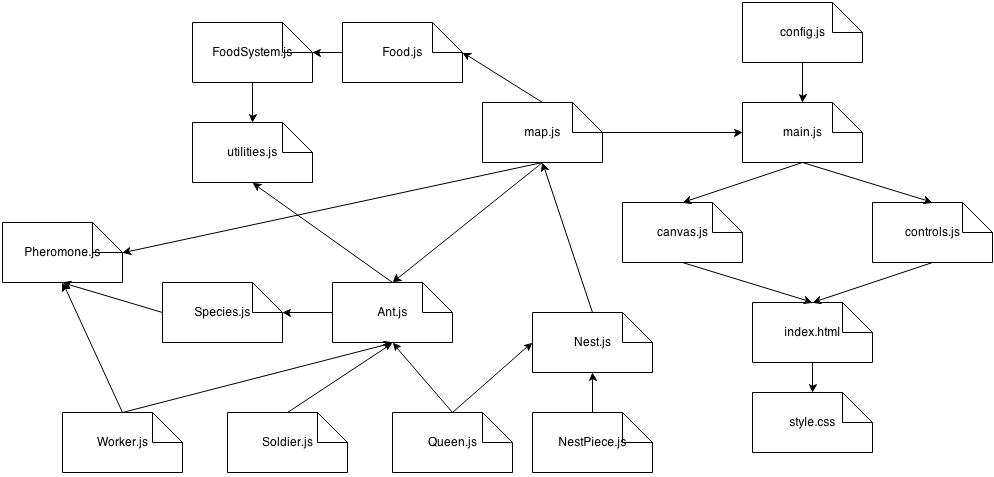
Maintenance

Contents

[System overview 3](#_Toc382483827)

[Modular structure overview 3](#_Toc382483828)

[ 4](#_Toc382483829)

[Index.html - Website structure 4](#_Toc382483830)

[Style.css - Page styling 5](#_Toc382483831)

[Core system 5](#_Toc382483832)

[Ants.js – The Ant class 6](#_Toc382483833)

[Worker.js – The Worker class 6](#_Toc382483834)

[Soldier.js – The Soldier class 6](#_Toc382483835)

[Queen.js – the Queen class 6](#_Toc382483836)

[Species.js – The Species class 6](#_Toc382483837)

[Nest.js – The Nest class 6](#_Toc382483838)

[NestPiece.js – The NestPiece class 6](#_Toc382483839)

[Pheromone.js – The Pheromone class 6](#_Toc382483840)

[FoodSystem.js – The FoodSystem class 7](#_Toc382483841)

[Food.js – The Food class 7](#_Toc382483842)

[map.js – The map module 7](#_Toc382483843)

[controls.js – The controls module 7](#_Toc382483844)

[8](#_Toc382483845)

[canvas.js – The canvas module 9](#_Toc382483846)

[utilities.js – The utilities module 9](#_Toc382483847)

[config.js – The configuration file 9](#_Toc382483848)

[main.js – The main program 9](#_Toc382483849)

[Tests 9](#_Toc382483850)

[Classes 10](#_Toc382483851)

[Ant 10](#_Toc382483852)

[Worker 12](#_Toc382483853)

[Queen 13](#_Toc382483854)

[Soldier 14](#_Toc382483855)

[Species 16](#_Toc382483856)

[Nest 18](#_Toc382483857)

[NestPiece 20](#_Toc382483858)

[Pheromone 20](#_Toc382483859)

[FoodSystem 21](#_Toc382483860)

[Food 21](#_Toc382483861)

[Modules 22](#_Toc382483862)

[Map 22](#_Toc382483863)

[Canvas 22](#_Toc382483864)

[Utilities 24](#_Toc382483865)

[Controls 26](#_Toc382483866)

[Main 31](#_Toc382483867)

[Config 33](#_Toc382483868)

[Difficult to understand code 37](#_Toc382483869)

[The addToMap function 37](#_Toc382483870)

[The Ant.wonder function 37](#_Toc382483871)

[The die function 38](#_Toc382483872)

[Food vs. Health 39](#_Toc382483873)

[Nest reproducing health 39](#_Toc382483874)

[The reproduce function 39](#_Toc382483875)

[Difference between Nest and NestPiece classes 40](#_Toc382483876)

[Goals 40](#_Toc382483877)

[The steps variable 40](#_Toc382483878)

[The guardNest function 40](#_Toc382483879)

[The mutate function 41](#_Toc382483880)

[this.sleep variable 42](#_Toc382483881)

[Configuration 42](#_Toc382483882)

[Simulation 42](#_Toc382483883)

[Environment 42](#_Toc382483884)

[Multipliers 43](#_Toc382483885)

[Global characteristics 43](#_Toc382483886)

[Species characteristics 43](#_Toc382483887)

[Interface 44](#_Toc382483888)

[Colours 44](#_Toc382483889)

[Size 44](#_Toc382483890)

[Controls 44](#_Toc382483891)

[Test driven development 44](#_Toc382483892)

[Run a test 44](#_Toc382483893)

[A passed test 45](#_Toc382483894)

[A failed test 45](#_Toc382483895)

[Human input 45](#_Toc382483896)

# System overview

The system created allows a user to see how the behaviour of ants is effected by their characteristics as well as editing these characteristics to design specific species. It is a single page web application which uses the canvas (introduced in the HTML5 specification) for rendering all shapes. The simulation implements three different types of ants:

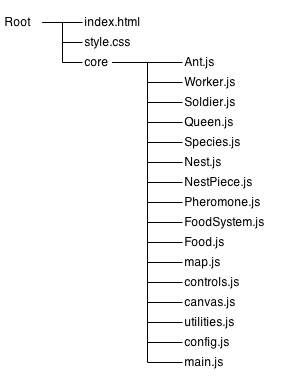
* **Worker ants** – These ants find and collect food to feed their nest
* **Soldier ants** – These ants guard their nests as well as the worker ants and food sources. They are also responsible for attacking other species of ants.
* **Queen ants** – These are responsible creating new nests

As well as a simple ant nests which allows the creation of new ants and acts as a store for deposited food. The simulation also includes food, food is used by ants to survive and in particular used by workers who collect it and deposit it at the nest. Food can re grow over time. Pheromones are also implemented in the form of worker ant food trials.

In the system the user has the ability to view how ants behave and evolve over time. The user can also change characteristics of ants to create a user defined species. As well as move around the simulation, zoom in and out and reset the simulation.

# Modular structure overview

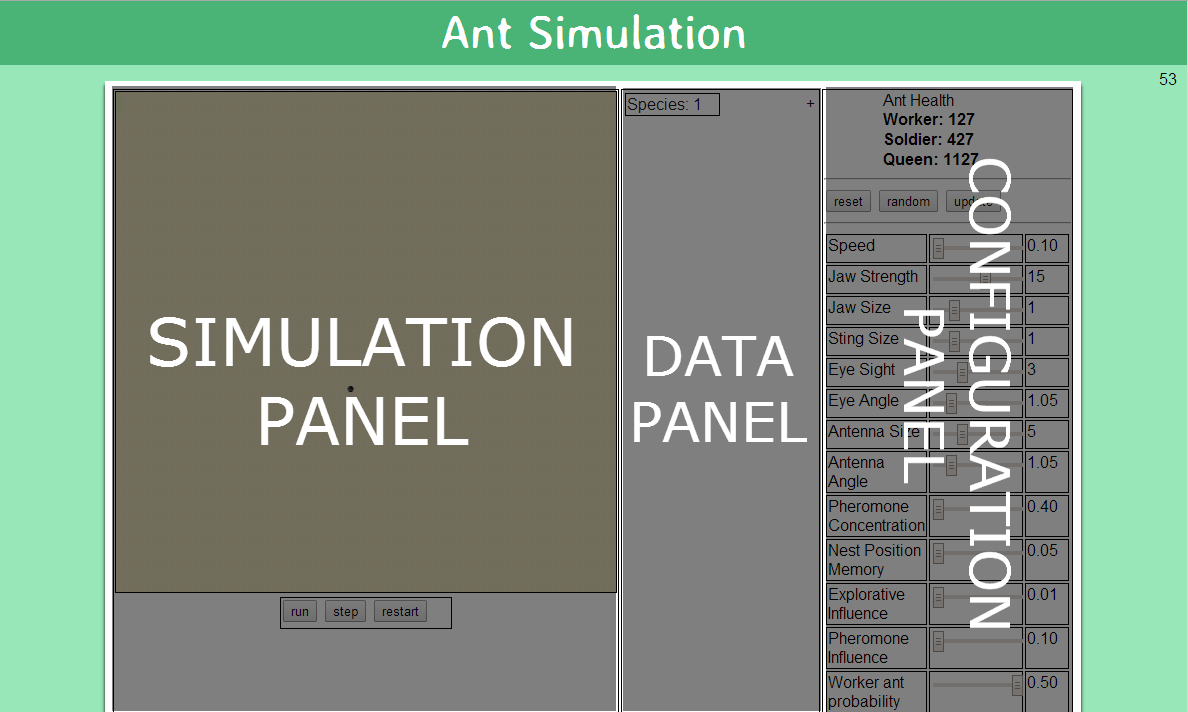
The applications files are split up into logical groups of code. Each group controls a single part of the system.



## C:\Dropbox\projects\Ant-Simulation\writeup\assests\Design\file structure\component diagram.png

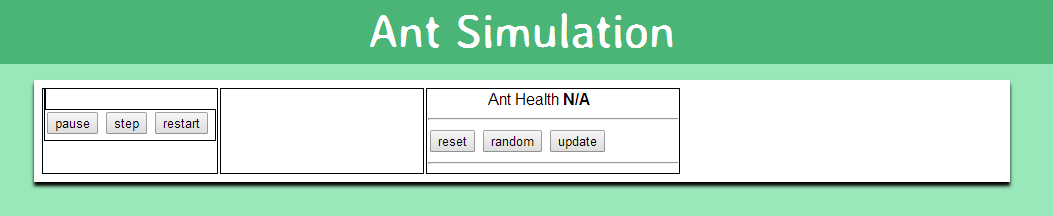
## Index.html - Website structure

The application is a single page website. The page is arranged into three sections:



* **The simulation panel** – Contains the simulation itself as well as controls for the simulation
* **The data panel** – Contains information about the different species in the simulation such as how many ants the species currently has.
* **The configuration panel** – This contains inputs which allow the user to change the values of characteristics for the currently selected species.

The page is written in HTML. The configuration panel is completely populated at run time with characteristics. This is done to more easily allow characteristics to be added to the system as they are only required to be added to core/config.js. The same is true for the data panel. Species data is populated throughout the simulation as species change. This means that the HTML code in index.html is only a small part of the actual HTML displayed in the system. The following is the page with scripts turned off, to demonstrate how much of the page is generated dynamically at run time.



## Style.css - Page styling

The page is styled in CSS *Note*: This styling only effects the HTML generated in the page. The simulation itself uses the HTML canvas API and is styled separately see configuration for styling the canvas.

## Core system

The application is split into 10 classes and 5 modules.

### Ants.js – The Ant class

The Ant class is the parent class of all other ant classes in the system (worker, soldier and queen). It defines the basic properties of an ant (size, position, id, species, home nest…) and its basic actions (takeFood, getFood, scan, smell…).

### Worker.js – The Worker class

The Worker class represents a worker ant. It defines properties and methods to do with the search, collection and deposition of food.

### Soldier.js – The Soldier class

The Soldier class represents a soldier ant. It defines properties and methods to do with the guarding and attacking of other ants and nests within the Map.

### Queen.js – the Queen class

The Queen class represents a queen ant. It defines properties and methods to do with the location and creation of a new nest.

### Species.js – The Species class

The Species class represents a species in the simulation. A species is a collection of specific characteristic values. The species class defines properties and methods to do with the storing and mutation of characteristics. All ants and nests have a species. Ants born from a particular nest will inherit the species of the nest. However if the ant is a queen ant, there is a chance that the species will be mutated and the queen will inherit the mutated species.

If a species is mutated, it means that a single characteristics value has been altered from its original value.

In the simulation, each species has a unique colour. The colour is randomly generated when the species is created. All ants and nests belonging to the species will also appear to be this colour.

For data collection purposes, the species class keeps track of all ants and nests which belong it its species. This is used in the data panel where a species data is shown.

### Nest.js – The Nest class

The Nest class represents an ant’s nest. It defines properties and methods to do with the creation of new ants.

### NestPiece.js – The NestPiece class

The NestPiece class represents a piece of the nest on the map. The nest which appears on the map is made up of a number of nest pieces, one per cell. This is done so that when a nest is attacked, parts of the nest can be destroyed while the rest of the nest remains. The nest pieces do not control any actions to do with the running of the nest and are simply physical representations of a nest object on the map. The Nest class on the other hand is the entity which controls the nest. Once all of a nests nest pieces are destroyed the nest will be destroyed. *Note*: See Nest vs. NestPieces in difficult to understand code.

### Pheromone.js – The Pheromone class

The Pheromone class represents a pheromone on the map. It defines methods and properties to do with adding, removing and updating a pheromone. A pheromone has a concentration, its concentration will slowly decay as it evaporates over time. When a pheromones concentration is <= 0 the pheromone is removed.

### FoodSystem.js – The FoodSystem class

The FoodSystem class defines properties and methods to place and grow food in the map. *Note*: The amount of food placed in the map is controlled in the config.js file. See the Environment section in configuration for details on how to edit the amount of food or how it grows.

The food

### Food.js – The Food class

The Food class represents a single piece of food. It defines properties and methods to place, draw and grow food. A piece of food contains an amount property, this is the amount of food that the piece contains. Different pieces of food can contain different amounts/concentrations of food, this is controlled by the FoodSystem object which created the food. Food of different concentrations appear different shades of the FOOD\_COLOUR. A darker shade means the food piece contains a higher concentration of food. While a lighter colour contains a lower concentration of food.

### map.js – The map module

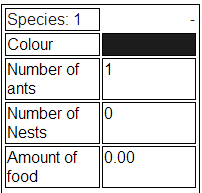
The map module defines functions relating to the creation, drawing and navigation of the map. The MAP itself is a global variable which represents the map which entitles are placed in in the simulation. The map is comprised of cells, each cell can contain multiple entities:

* MAP[index].ants – Is an array of all the ants in the current cell
* MAP[index].pheromones – Is an array of all the pheromones in the cell
* MAP[index].food – Contains either void(0) if there is no food in the cell or a food object if there is.

*Note*: Ants can move distances smaller than the size of a single cell, this means that ants may exist between cells when drawn. However on the MAP the cell which the ant lies in mostly will be used as the cell which the ant is in.

### controls.js – The controls module

The controls module is responsible for HTML generation. It controls functions for the creation and removal of HTML elements. The module is used when the page is first loaded to resize the canvas and generate the inputs in the configuration panel from the characteristics defined in config.js. It is then used to generate species data when new species are created, and update species data each tick.

The following are examples of some of the HTML generated by the controls module:

### C:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\HTML generation\Generated HTML configuration panel - referes to.PNGC:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\HTML generation\Generated HTML configuration panel.PNG

### canvas.js – The canvas module

The canvas module contains a collection of functions which operate on the canvas as well as use the canvas API. The canvas is tag in the HTML 5 specification, it allows for dynamic, scriptable rendering of 2D shapes. The module mainly contains shortcuts for drawing shapes such as rectangles, lines, arcs and circles. However also contains functions for resizing and clearing the canvas.

*Note*: The canvas module is the only location where calls to the canvas API are used. This is done to increase maintenance as if the specification changes or is updated, only a single modules must be changed to reflect the changes. And so making it easier to maintain.

### utilities.js – The utilities module

The utilities module contains a collection of functions which are used by multiple classes and modules. It contains functions for generation of random values, calculation of distances, angles and boundaries, coordinate manipulating and conversion, retrieving selections of cells from the MAP and creation of ants.

### config.js – The configuration file

The configuration file contains all of the system configurable values. Configuration for the entire system is done using this file. This is to improve maintainability as it means only a single file must be changed and it also means it is easier to locate configuration options.

### main.js – The main program

This file is contains the main update loop for the simulation, and creates the simulation environment as well as defying event listeners for events such as shortcut key presses. The main update loop loops through every cell in the map and updates every pheromone, piece of food and ant each tick. The environment created creates a FoodSystem, generates food and adds single queen ant to start the simulation.

## Tests

The system contains a large number of unit tests for all parts of the system. See test driven development section for more details.

# Classes

*Note*: parameters of the function are appended with **[param]**

|  |  |
| --- | --- |
| **Type** | **Description** |
| integer | A whole number |
| number | A real number |
| string | A list of characters |
| Boolean | A true/1 or false/0 |
| \* object | An object |

*Note*: As JavaScript is weakly type the types stated are the intended types / how they are used.

## Ant

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Ant | this.size | width : integer, height : integer | The size of the ant in pixels (default: CELL\_SIZE) |
| this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.species | Species object | The ants species which determines its characteristics |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker |
| this.nest | Nest object | The ants home nest where it was born |
| this.colour | string | The hexadecimal colour of the ant e.g. ‘#FF0000’ (default: ‘#1C1C1C’) |
| this.health | number | The health of the ant, if <= 0 the ant is dead |
| this.hungerThreshold | number | The value of health bellow which the ant is determined to be hungry (default: 100) |
| this.healthRate | number | The rate at which the ants health decreases per tick (default: 0.1) |
| this.alive | boolean | Represents the ants living state (default: true) |
| this.goal | integer | The current goal which the ant is trying to accomplish, use the GOAL object when setting e.g. GOAL.findFood (default: GOAL.none) |
| this.target | x : number, y : number | The coordinate of a target the ant has chosen, the type of target depends on the ant i.e. a worker ant targets food while soldier ants target enemy ants (default: void(0)) |
| this.itemsInView | ants: [Ant object], food : [Food object] | Holds arrays of all ants and food within view (default: ants: [], food: []) |
| this.pheromonesInRange | [Pheromone object] | An array of all the pheromones in the ants antenna range (default: []) |
| this.sleep | integer | The number of ticks the ant needs to sleep for. Used for tasks which require actions which take multiple ticks to complete (default: 0) |
| this.followingPheromone | boolean | Used to tell if an ant is following a pheromone (default: false) |
| this.direction | number | The angle which the ant is facing in radians from the vertical axis clockwise (default: \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| Ant.addToMap | N/A |  | Adds the current position of the ant on the map |
| Ant.removeFromMap | N/A |  | Removes the current position of the ant from the map |
| Ant.isHungry | N/A |  | Determines whether the ant is hungry or not.  **Return** boolean - true if the ant is hungry else false |
| Ant.updateSleep | N/A |  | Updates this.sleep variable to simulate time passing during sleep |
| Ant.isFood | **[param]** food | Food object | Determines whether a piece of food exists or not  **Return** boolean - true if piece of food exists else false |
| Ant.takeFood | **[param]** food | Food.object | Takes a single piece of food  **Return** boolean - true if there is still food left, else false |
| Ant.atNest | N/A |  | Determines whether the ant is currently at its own nest i.e. standing on top of a NestPiece  **Return** boolean - true if the ant is standing on its nest, else false |
| Ant.seeNest | N/A |  | Determines whether the ant can see its own nest  **Return** boolean - true if ant can see its nest, else false |
| Ant.findFoodTarget | N/A |  | Choose the target piece of food the ant should go for |
| Ant.getFood | N/A |  | Walk towards food until on top of it and then pick it up one piece at a time |
| Ant.useFood | N/A |  | Determines the best use of food i.e. either eating the food if hungry or carrying it |
| Ant.scan | N/A |  | Looks at all blocks in front of the ant within eyesight, places all items of interest into this.itemsInView |
| Ant.smell | N/A |  | Similar to this.scan - Looks at all blocks in front of the ant within antenna size, places all pheromones into this.pheromoensInRange |
| Ant.secrete | N/A |  | Secrete pheromones |
| Ant.wonder | N/A |  | Wonder around the map, following pheromones of own species otherwise picking random directions |
| Ant.move | N/A |  | Updates the ants coordinates |
| Ant.die | N/A |  | Removes the ant from the simulation |

## Worker

Inherits Ant

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Worker | this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.worker) |
| this.direction | number | The angle which the ant is facing in radians from the vertical axis clockwise (default: \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| this.carrying | integer | The amount of food the ant is carrying (default: 0) |
| this.carryingThreshold | integer | If an ant is carrying more food then this value and cannot see any food near it, ant will return to the nest to deposit the food (default: 4) |
| Worker.canCarry | N/A |  | Determines if an ant can carry food or not  **Return** boolean - true if ant can carry food else false |
| Worker.depositeFood | N/A |  | Navigate towards the nest and deposit food at the nest |
| Worker.dropFood | N/A |  | Drop a single piece of food at the nest  **Return** boolean - True if there is no more food to drop off, else false |
| Worker.useFood | N/A |  | Determines the best use of food i.e. eating it if hungry or carrying it |
|  | index - The MAP index of the ant | integer |
|  | food - The food object which is being used | Food object |
| Worker.doTask | N/A |  | Performs the actions required to complete a task |
| Worker.updateGoal | N/A |  | Determines if a goal has been completed or not and updates the next goal for the ant |
| Worker.updateHealth | N/A |  | Updates the ants this.health variable. Differs from ant.updateHealth as allows ants to eat food they are carrying if hungry |
| Worker.draw | **[param]** ctx | Canvas context object | Draws the ant onto the canvas context |
|  | scaledCoord - The coordinate of the map scaled to pixels | integer |
| Worker.update | N/A |  | Update the ant each tick |

## Queen

Inherits Ant

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Queen | this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.queen) |
| this.steps | integer | The number of steps the queen will take until reaching the nest site |
| Queen.doTask | N/A |  | Decide what actions need to be done to accomplish a task |
| Queen.updateGoal | N/A |  | Checks to see if the goal is accomplished and updates it if necessary |
| Queen.pickDirection | N/A |  | Used to pick a direction in which the Queen will walk a specific number of steps in (this.steps) and then create a nest |
| Queen.createNest | N/A |  | Creates a nest object |
|  | nest - The MAP index of the ant | Nest object |
|  | index - The index of the queen ant in this.species.ants used to determine which ant to remove from the list | integer |
| Queen.draw | **[param]** ctx – The canvas context | Canvas context object | Draw the queen onto the canvas context |
|  | scaledCoord - The scaled MAP coordinate to pixels | integer |
| Queen.update | N/A |  | Update the queen each tick |

## Soldier

Inherits Ant

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Soldier | this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.soldier) |
| this.direction | number | The direction in radians from the vertical axis clockwise (default \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| this.targetAnt | Ant object | The enemy ant which the soldier is targeting to attack (default: void(0)) |
| this.moving | boolean | Used for determining when an ant is in a static position e.g. guarding the nest (default: false) |
| this.steps | integer | Used for moving an ant a certain number of steps away from an object. |
| this.nearNest | boolean | Determine if an ant is close to the nest (even if it is not in view) (default: false) |
| this.nearFood | boolean | Determine if an ant is close to food (even if it is not in view) (default: false) |
| Soldier.soldiersInView | N/A |  | Determines if there are other friendly soldiers in view  **Return** boolean – true if there are soldiers in view else false |
| Soldier.seeFood | N/A |  | Determines if an ant can see food  **Return** boolean – true if the ant can see food else false |
| Soldier.pickTarget | N/A |  | Chooses which ant the soldier should target |
| Soldier.follow | N/A |  | Sets the ants direction to intercept the path of the targeted ant so that it can attack. |
| Soldier.attack | N/A |  | Attack a the targeted ant if in range |
|  | dist - The distance between the ant and its target | number |
| Soldier.updateHealth | N/A |  | Updates the soldier ants health differs from Ant.updateHealth as will change soldiers goal if hungry |
| Soldier.updateSteps | N/A |  | Updates the this.steps variable |
| Soldier.guardNest | N/A |  | Controls soldiers logic if assigned goal of guarding the nest |
| Soldier.guardPheromone | N/A |  | Controls soldiers logic if assigned goal of guarding pheromone trials |
| Soldier.guardFood | N/A |  | Controls soldiers logic if assigned goal of guarding food |
| Soldier.findFood | N/A |  | Controls soldiers logic if assigned goal of finding food e.g. if the ant is hungry |
| Soldier.doTask | N/A |  | Performs the actions required to complete a task |
| Soldier.updateGoal | N/A |  | Determines if a goal has been completed or not and updates the next goal for the ant |
| Soldier.draw | **[param]** ctx – The canvas context | Canvas context object | Draw the ant onto the canvas context |
| scaledCoord - The scaled MAP coordinate to pixels | integer |
| Soldier.update | N/A |  | Update the ant each tick |

## Species

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Species | this.id | integer | The unique ant id |
| this.ants | [Ant object] | Array of all ants which belong to the species (default: []) |
| this.nests | [Nest object] | Array of all nests which belong to the species (default: []) |
| this.chars.speed | number | The speed the ant moves, 1 = 1 cell per tick (default: 0.25) |
| this.chars.antennaSize | integer | The range the ant can smell pheromones (default: 5) |
| this.chars.jawStrength | integer | The strength of the ants jaw (determines how much food the ant can carry) (default: 10) |
| this.chars.jawSize | integer | The amount of damage a soldier ant does when attacking (default: 1) |
| this.chars.stingSize | integer | The range which the solider ant can attack (default: 1) |
| this.chars.eyeSight | integer | The range the ant can see items In front of it (in number of cells) (default: 5) |
| this.chars.eyeAngle | number | The angle of the sector the ant can see in front of it (default: π/2) |
| this.chars.antennaAngle | number | the angle of the sector the can can smell pheromones in front of it (default: π/2) |
| this.chars.pheromoneConcentration | number | The concentration of pheromones secreted (default: 0.4) |
| this.chars.nestCoordMemory | number | A measure of how well the ant knows where the nest is, used when navigating to the nest, represents memory of familiar landmarks near the nest (default: 0.1) |
| this.chars.explorativeInfluence | number | The likelihood of an ant changing direction rather than continuing going the direction its facing (default: 0.05) |
| this.chars.pheromoneInfluence | number | How likely it is that an ant will follow a pheromone (default: 0.9) |
| this.chars.queenStepsMin, this.chars.queenStepsMax | number | The range of steps the queen will take when navigating to a new nest site i.e. lower values mean closer nests (default: min: 200, max: 800) |
| this.chars.reproduceWorkerProb, this.chars.reproduceQueenProb, this.chars.reproduceSoldierProb | number | The probability a particular type of ant will be born compared with others (default: worker: 0.5, queen: 0.05, soldier: 0.1) Note: gets normalized so does not need to add to one. |
| this.chars.reproduceWorkerFoodCost, this.chars.reproduceQueenFoodCost, this.chars.reproduceSoldierFoodCost | number | The amount of food required to create this type of ant (This is the amount of food the ant will start with when born) (default: worker: 5, queen: 25, solider: 8) |
| this.chars.reprodcutionRate | number | The chance each tick of creating a new ant (default: 0.05) |
| this.colour.worker, this.colour.soldier, this.colour.queen, this.colour.nest, this.colour.pheromone | string | The hexadecimal colours given to each type of ant/nest/pheromone. Note: often all values are the same, (default: ‘#1C1C1C’ for all) |
| this.murationRate | number | The chance that a characteristic will be mutated |
| Species.mutateChar | **[param]** characteristic | string | Mutate a specific characteristic  **Return** number – The mutated value of the characteristic |
| Species.mutate | N/A |  | Mutate a single characteristic in the species  **Return** Species object – The new mutated species |
| altChars - The current characteristics of the species | object literal |
| altCharacterisitc - The random characteristic which will be mutated | string |
| altValue - The new value of the altCharacterisitc | variable depending on characteristic being mutated |
| species - The species new species | Species object |
| Species.createSpecies | **[param]** chars | object literal | The characteristic set of the new species  **return** Species object – The new species |
| species - The new species | Species object |
| colour - The colour of the new species | string |

## Nest

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **Nest** | this.nestSize | x: integer, y: integer | The number of pieces the nest extends in both the x and the y directions (default: NEST\_SIZE) |
| this.coord | x: number, y: number | The coordinate of the nest |
| this.id | integer | A unique identifier |
| this.species | Species object | The species which the nest belongs to |
| this.pieces | [NestPiece object] | An array of the NestPiece objects belonging to the nest (default: []) |
| this.health | number | The health the nest has (default: 10000) |
| this.hungerThreshold | number | The threshold below which the nest is hungry and tries to preserve food (default: 1000) |
| this.healthRate | number | The rate at which the nests health reduces each tick (default: 0.1) |
| this.alive | boolean | If nest is alive or not, needed if nest dies mid execution so does not keep acting as if it is alive for the rest of the tick (default: true) |
| **Nest.addNestPiece** | [param] coord | x: number, y: number | Create a single nest piece |
| **Nest.createNest** | N/A |  | Creates all the nest pieces |
| **Nest.die** | N/A |  | Kills the nest so that it is no longer updated and is removed from the map |
| **Nest.getCost** | [param] type – The ant type | integer – use the ANT\_TYPE object when defining | Returns the cost in the amount of health needed to create a specific type of ant  Return number – the health required |
| **Nest.calcSpeciesCost** | N/A |  | Calculates the cost of the characteristics due the spices  Return number – the health cost |
| **Nest.viable** | [param] type – The ant type | integer – use the ANT\_TYPE object when defining | Determine whether or not it is viable to create a specific type of ant  Return boolean – true if it is viable else false |
| **Nest.createAnt** | [param] type – the ant type | integer – use the ANT\_TYPE object when defining | Create a new ant |
| cost - The health cost of creating the type of ant | number |
| **Nest.reproduce** | N/A |  | Determines what type of ant to create |
| prob - A random number which will be used to determine what type of ant is created | number |
| chars - The characteristics of the species | number |
| sum - The sum of ant probabilities | number |
| queenProb, soldierProb, workerProb - The respective normalized probabilities of each type of ant | number |
| ordered - The ordered normalized probabilities of each type of ant being created | [{prob : number, type: integer}] |
| **Nest.updateHealth** | N/A |  | Update the nests health |
| **Nest.update** | N/A |  | Updates the nest each tick |

## NestPiece

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **NestPiece** | this.size | width: number, height: number | The size the piece will be in pixels (default: CELL\_SIZE) |
| this.coord | x: number, y: number | The coordinate of the piece |
| this.nest | Nest object | The nest object which controls the piece |
| this.id | integer | A unique identifier |
| this.type | integer – use the ANT\_TYPE object when defining | The type of object the piece is, as the nest piece is essentially treated as a static ant and is stored under ant in MAP, the type is needed so ants can identify the piece (default: ANT\_TYPE.nest) |
| this.heatlh | number | The health the piece has |
| **NestPiece.addToMap** | N/A |  | Adds the piece to the map |
| **Nest.removeFromMap** | N/A |  | Removes the piece from the map |
| **Nest.die** | N/A |  | Kills the nest piece stopping it from being updated and removes it from the map |
| **Nest.draw** | [param] ctx | Canvas context object | Draws the nest piece onto the canvas context |

## Pheromone

|  |  |  |  |
| --- | --- | --- | --- |
| **Function name** | **Variables** | **Type** | **Description** |
| **Pheromone** | this.concentration | number | The concentration of the pheromone |
| this.coord | x: number, y: number | The coordinate of the pheromone |
| this.size | x: integer, y: integer | the size of the pheromone in pixels (default: CELL\_SIZE) |
| this.species | Species object | The species which the pheromone belongs to |
| **Pheromone.addToMap** | N/A |  | Adds the pheromone to the map |
| **Pheromone.removeFromMap** | N/A |  | Removes the pheromone from the map |
| **Pheromone.draw** | [param] ctx – The canvas context | Canvas context object | Draw the pheromone onto the canvas context |
| **Pheromone.update** | N/A |  | Updates the pheromones concentration each tick |

## FoodSystem

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **FoodSystem** | this.variation | min: integer, max: integer | The minimum and maximum amounts of food a single piece can contain |
|  | this.colour | string | The hexadecimal colour of the piece of food (default: FOOD\_COLOUR). Note: opacity depends on food amount |
| **FoodSystem.addFoodBlob** | [param] coord – The coordinate of the food | x: number, y: number | Creates a circle of food with reducing amounts around a coordinate. |
|  | [param] radius – The radius of the food | integer |
|  | affectedCells – The cells which lie in the food blob | [{x: integer, y: integer}] |
|  | distanceFromCenter – The distance between the cell and the coord | number |
|  | amount – The food amount | integer |
| **FoodSystem.addFood** | N/A |  | Adds random sized food blobs at random positions in the map |

## Food

|  |  |  |  |
| --- | --- | --- | --- |
| **Function name** | **Variables** | **Type** | **Description** |
| **Food** | this.foodSystem | FoodSystem object | The food system which controls the food pieces |
|  | this.size | width: number, height: number | The size of the food in pixels (default: CELL\_SIZE) |
|  | this.amount | integer | The concentration of food the piece has |
|  | this.coord | x: number, y: number | The coordinate of the piece of food |
| **Food.addToMap** | N/A |  | Adds the piece of food to the map |
| **Food.removeFromMap** | N/A |  | Removes the piece of food from the map |
| **Food.draw** | [param] ctx – The canvas context | Canvas context object | Draws the piece of food onto the canvas context |

# Modules

## Map

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **drawGrid** | [param] ctx | Canvas context object | Draws a grid onto the canvas context |
| **drawBackground** | [param] ctx | Canvas context object | Draws a rectangle over the entire canvas effectively wiping it to a single colour |
| **zoom** | [param] level – The zoom level (positive : zoom in, negative : zoom out) | number | Change the simulation zoom level |
| **createMap** | N/A |  | Create an empty map and populate it with default values |

## Canvas

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **resizeElement** | [param] element – The element to be resized | HTML element | Resizes a HTML element to a size |
| [param] size – The new size of the element | width: number, height: number |
| **drawRect** | [param] ctx – The canvas context | Canvas context object | Draws a rectangle onto a canvas context |
| [param] coord – The coordinate of the top left corner | x: number, y: number |
| [param] size – The size of the rectable | width: number, height: number |
| [param] fillColour – The colour of the rectangle | string |
| [param] strokeColour – The colour of the rectangles boarder (default: ‘#000000’) | string |
| [param] lineWidth – The width of the border (default: 0) | number |
| **drawLine** | [param] ctx – The context which the line will be drawn onto | Canvas context object | Draws a line onto a canvas context |
| [param] coord1 – The coordinate of the starting point | x: number, y: number |
| [param] coord2 – The coordinate of the end point | x: number, y: number |
| [param] strokeColour – The stroke colour of the line | string |
| [param] lineWidth – The width of the stroke | number |
| **drawArc** | [param] ctx – The canvas context which the arc will be drawn onto | Canvas context object | Draws an arc onto a canvas context |
| [param] coord – The coordinate of the centre of the arc | x: number, y: number |
| [param] radius – The radius of the arc | number |
| [param] startAngle – The angle from the horizontal to start the arc at (clockwise) | number |
| [param] endAngle – The angle from the horizontal to stop the arc at (clockwise) | number |
| [param] strokeColour – The stroke colour of the arc | string |
| [param] lineWidth – The width of the stroke | number |
| [param] fillColour – The colour of the arc | string |
| **drawCircle** | [param] ctx – The canvas context which the circle will be drawn onto | Canvas context object | Draws a circle onto a canvas context |
| [param] coord – The coordinate of the centre of the circle | x: number, y: number |  |
| [param] radius – The radius of the circle | number |  |
| [param] fillColour – The colour of the circle | string |  |
| **clearCanvas** | [param] ctx – The canvas context | Canvas context object | Draws a rectangle over the entire canvas effectively wiping it to a single colour (OUT\_OF\_BOUNDS\_COLOUR) |

## Utilities

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **randInt** | [param] range – The range inclusive | min: integer, max: integer | Returns a random integer within a specific range  Return integer – The random number |
| **randFloat** | [param] range – The range inclusive | min: number, max: number | Returns a random float within a specific range  Return number – The random float |
| **randDir** | N/A |  | Returns a random angle between 0 and 2π radians  Return number – The random direction |
| **randColour** | N/A |  | Returns a random hexadecimal colour  Return string – The random colour |
| **validateDirection** | [param] dir – The direction | number | Returns an angle in the range 0 to 2π e.g. if dir = 4π returns 2π  Return number – The new angle |
| **randProperty** | [param] obj – The object literal | object | Picks a random property of an object literal  Return object – The random property from the object |
| **distance** | [param] coord1, coord2 – The coordinates to find the distance between | x: number, y: number | Returns the shortest distance between two coordinates  Return number – The distance |
| x – The distance in the x axis | number |
| y – The distance in the y axis | number |
| **scaleCoord** | [param] coord – The coordinate to scale | x: number, y: number | Scale a coordinate to its location in pixels  Return {x: number, y: number} – The scaled coordinate |
| **coordToIndex** | [param] coord – The coordinate which will be converted | x: number, y: number | Converts a coordinate to a map index, it can be used with coordinates which don’t map exactly to a single index and will work out which cell the coordinate lies in mostly  Return integer – The map index which the coordinate converts to |
| **indexToCoord** | [param] index – The map index | integer | Converts a map index to a coordinate  Return {x: number, y: number} – The coordinate which the index converts to |
| x – The x coordinate of the index | number |
| y – The y coordinate of the index | number |
| **getCellCoord** | [param] coord – The coordinate | x: number, y: number | Similar to coordToIndex however returns the \*neat\* coordinate i.e. a coordinate which maps exactly to a single map index by finding the cell which the coordinate lies in mostly  Return {x: number, y: number} – The coordinate which maps exactly to a single map index |
| **boundary** | [param] coord – The coordinate to test | x: number, y: number | Returns the \*wrapped\* coordinate of a coordinate if it exceeds the map boundary. As the map wraps around if an ant foes off one side it will appear on the other  Return {x: number, y: number} – The wrapped coordinate |
| [param] bounds – The boundary which is tested against | x: {min: number, max: number}, y: {min: number, max: number} |
| **getBlock** | [param] coord – The coordinate to get the block around | x: number, y: number | Returns an array of cells which lie a certain distance around a specific point  Return {x: number, y: number} – An array of coordinate which lie around the coordinate |
| [param] size – The size of the block i.e. if width = 2, takes 2 blocks to the left and two blocks on the right of the coordinate | width: integer, height: integer |
| **getSector** | [param] coord – The coordinate to get the block around | x: number, y: number | Returns an array of cells which lie in the sector of a circle of a particular radius around a specific point  Return [{x: number, y: number}] – An array of coordinates which lie in the sector |
| [param] radius – The radius of the sector | number |
| [param] direction – The direction the ant is facing | number |
| [param] angle – The angle of the sector | number |
| searchCoord – The coordinate of the cell currently being checked to see if it lies in the sector | x: number, y: number |
| **turnAround** | angle – The angle in radians | number | Returns the reverse direction  Return number – The reversed direction |
| **angleTo** | [param] coord, target – The coordinates to find the angle between | x: number, y: number | Returns the direction/angle of shortest path to get from the coord to the target  Return number – the angle from the vertical axis clockwise in radians |
| **createAnt** | [param] species – The species of the new ant | Species object | Creates a new ant |
| [param] coord – The coordinate of the new ant | x: number, y: number |
| [param] nest – The new ants home nest | Nest object |
| [param] startingHealth – The new ants health | number |
| [param] type – The type of ant to create e.g. ANT\_TYPE.worker | integer |
| **genID** | N/A |  | Generates a unique id. Requires CURRENT\_ID variable to keep track of current id  Return integer – A unique ID |
| **clone** | [param] obj – The object which will be cloned | object | Clones an object (needed as JavaScript passes everything by reference)  Return object – A copy of object |
| temp – A copy of the object | object |
| **getElement** | [param] id – The HTML id | string | Returns the HTML element on ID responds to  Return HTML element object – The HTML element |
| **setValue** | [param] id – The HTML id | string | Sets the value of a HTML element |
| [param] value – The value which the HTML element is being set to | string |

## Controls

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **start** | N/A |  | Starts the simulation |
| **pause** | N/A |  | Stops the simulation |
| **toggleRunning** | N/A |  | Toggles between running and paused |
| **step** | N/A |  | Steps through a single tick of the simulation |
| **runPauseButton** | N/A |  | Toggles running and updates the pause/run button |
| **updateValue** | [param] element – An element of an input in the settings panel | HTML element object | Updates the value of a characteristic i.e. when a user slides a slider, the characteristics value is updated |
| [param] value – The new value of the characteristic | string |
| characteristic – The characteristic being updated | object literal |
| speciesCost – The cost of the species with the updated value | number |
| workerFoodCost, queenFoodCost, soldierFoodCost – The costs of each type of ant | number |
| queenStepsMax, queenStepsMin – The number of queen steps (used to determine if the min is > then the max) | number |
| **updateDefaultValues** | N/A |  | Updates the value of all characteristics of the selected species to their default values |
| characteristic – The characteristic being changed | object literal |
| **updateRandomValues** | N/A |  | Updates the value of all characteristics of the selected species to random values |
| characteristic – The characteristic being changed | object literal |
| value the new value of the characteritic | number |
| **newElement** | [param] tag – A HTML tag name e.g. ‘div’ | string | Creates a new HTML element  Return HTML element – The new element |
| [param] attribute – An array of attributes to add tot the new element | [{type: string, value: \*}] |
| element – The new HTML element |  |
| **createInputType** | [param] characteristic – A single characteristic from CHARS | object literal | Creates an input element e.g. a range input or a button input. |
| [param] prop – The property of CHARS which the characteristic refers to | string |
| input – the input element | HTML element |
| **createInput** | [param] characteristic – A single characteristic from CHARS | object literal | Creates a row containing a label, input and value elements used for creating dynamic inputs for characteristics  Return HTML element – The row containing the label, input and value |
| [param] prop – The property of CHARS which the characteristic refers to | string |
| row – The row in the table which will contain the characteristic | HTML element |
| label – The table item containing the characteristics label | HTML element |
| inputContainer – The container which will hold the input. Needed to scale the input correctly with css | HTML element |
| input – The input element | HTML element |
| value – The table item containing the current value of the input | HTML element |
| **createCharacterisitcInputs** | N/A |  | Creates inputs for all characteristics used for creating dynamic inputs for characteristics. These are all appended to a table in the configuration panel |
| configPanel – The div which holds all elements in the configuration panel | HTML element |
| table – The table holding all the characteristic | HTML element |
| inputRow – The row returned by createInput | HTML element |
| **updateUserSpecies** | N/A |  | Updates the selected species with the new values selected in the configuration panel i.e. what happens when the update button is pressed |
| speciesCost – The total cost of the species in health | number |
| characteristic – The current characteristic being updated | object literal |
| **createDataRow** | [param] className – The name of the class for the species | string | Creates a row containing a label and data elements used for displaying information about species  Return HTML element – The data row |
| [param] id – The ID which will be used to access and update the species | string |
| [param] labelValue – The text which will be used as a label for the data | string |
| [param] dataValue – The data which will be displayed | string |
| row – The HTML row element which will contain the label and data | HTML element |
| label – The table item which contains the labelValue text | HTML element |
| data – The table item which contains the dataValue text | HTML element |
| **createSpeciesData** | [param] species – The species whose data you want to create a data display for | Species object | Create all elements to display information about a species |
| id – The species ID | string |
| className – The name of the class which responds to all items within a species data | string |
| table – The table which will contain all of the data | HTML element |
| titleRow - the HTML row which contains the title | HTML element |
| title – The title showing which species the table refers to | HTML element |
| toggleVisibility – The button which is used to show and hide the species datat | HTML element |
| colourRow – The row containing information about the species colour | HTML element |
| antNumRow – The row containing information about the number of ants in the species | HTML element |
| nestNumRow – The row containing information about the number of nests in the species | HTML element |
| foodAmount – The amount of food all the nests in the species have combined | number |
| foodAmountRow – The row containg information about the amount of food the species has i.e. all of the nests combined | HTML element |
| button – Used to toggle the visibility of the data (this must be done once the table has been appended to the data panel) | HTML element |
| **updateSpeciesData** | N/A |  | Updates data for all of the species currently in the simulation i.e. so that the information in the data panel reflects the current state of the species in the simulation |
| species – The species being updated | Species object |
| id – The ID of that species | string |
| colourDataElement – The element which contains information about the species colour | HTML element |
| antNumDataElement - The element which contains information about the species number of ants | HTML element |
| nestNumDataElement - The element which contains information about the species number of nests | HTML element |
| foodAmountDataElement - The element which contains information about the species amount of food in all nests | HTML element |
| foodAmount – The amount of food contained in each nest in the species combined | number |
| **removeSpeciesData** | [param] id – The id of the species to remove | string | Removes a species data i.e. when it has died out |
| **nextVisibility** | [param] button – A reference to the toggle visibility button | HTML element | Checks whether the species data id currently maximised or minimised and returns whether or not the species data should be hidden or not assuming the button has been clicked  Return string – The value of the displaying styling |
| **toggleVisibilityButton** | [param] button – A reference to the toggle visibility button | HTML element | Toggles the text in the visibility button |
| **toggleClassVisibility** | [param] button – A reference to the toggle visibility button | HTML element | Maximises and Minimises the data for a particular species |
| id – The id of the species the toggle visibility button corresponds to | string |
| listOfClassElements – All elements of the same class i.e. all elements in the species data to be hidden | [HTML element] |
| **select** | [param] title – A reference to the title element of the species data | HTML element | Selects a new species allowing the user to alter that species characteristics |
| id – The id of the species | string |
| species – The species being selected | Species object |
| characteristic – The characteristic being updated | HTML element |

Main

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| **tick** | N/A |  | Performs actions required each tick |
| **drawMap** | [param] ctx – The canvas context | Canvas context object | Draws all objects onto the canvas and also updates all objects |
| ant – The ant to draw | Ant object |
| pheromone – The pheromone to draw | Pheromone object |
| food – The piece of food to draw | Food object |
| **createEnviroment** | N/A |  | Setups the environment when starting or restarting the simulation |
| simulationFoodSystem – The food system which controls the placement of food in the map | FoodSystem object |
| x – The random x-coordinate of the starting queen | integer |
|  | y – The random y-coordinate of the starting queen | integer |  |
| **window.onload (Anonymous function)** | N/A |  | Performs actions once the page is loaded i.e. all scripts are loaded and the canvas is loaded |
| start – The coordinate of the start of the drag | x: number, y: number |
| end – The coordinate of the end of the drag | x: number, y: number |
| **window.onkeydown (Anonymous function)** | [param] e – The event | Event object | A call back function for when a key down press event happens. Controls what each key does e.g. arrow keys pan |
| charCode – The character code the event maps to | string |
| **canvasDOM.addEventListener(‘mousewheel’) (Anonymous function)** | [param] e – The event | Event object | Controls scrolling in the map when the mouse wheel is used (Chrome and IE only) |
| delta – The amount scrolled | number |
| **canvasDOM.addEventListener(‘DOMMouseScroll’) (Anonymous function)** | [param] e – The event | Event object | Controls scrolling in the map when the mouse wheel is used (Firefox only) |
| delta – The amount scrolled | number |
| **canvasDOM.onmousedown (Anonymous function)** | [param] e – The event | Event object | Controls panning behaviour for when the mouse is down |
| **canvasDOM.touchstart (Anonymous function)** | [param] e – The event | Event object | Controls panning behaviour when the canvas is touched |
| **canvasDOM.onmousemove (Anonymous function)** | [param] e – The event | Event object | Controls panning behaviour when the mouse is being moved |
| **canvasDOM.touchMove (Anonymous function)** | [param] e – The event | Event object | Controls panning behaviour when a finger is moved across the simulation |
| **canvasDOM.mouseup (Anonymous function)** | [param] e – The event | Event object | Stops panning the simulation |
| **canvasDOM.touchend (Anonymous function)** | [param] e – The event | Event object | Stops panning the simulation |

|  |  |  |
| --- | --- | --- |
| **Global variable** | **Type** | **Description** |
| **canvasDOM** | HTML element | The canvas HTML element |
| **canvasCTX** | Canvas context object | The canvas elements context (used for drawing to) |
| **lastLoop** | number | The time taken for the last tick to complete in milliseconds |
| **BUTTONS** | object literal | Holds the actions that buttons do e.g. when the reset button is pressed (must have id = ‘reset’) call the updateDefaultValues function would be BUTTONS = {reset: updateDefaultValues} |
| **draging** | boolean | Holds the current state of the simulations pan i.e. if the simulation is currently being dragged by a mouse or finger. (default: false) |

Config

| **Global variable** | **Type** | **Description** |
| --- | --- | --- |
| **VALUE\_TYPE** | object literal (constant) | Used to signify if a variable is an integer or a float i.e. needed as some characteristics require integer values while others require floats and so to generate random values for these characteristics must know what type they need. |
| **TICK** | number (constant) | The time between ticks (default: 10) |
| **CURRENT\_ID** | number | Used to keep track of the latest unique ID (default: 0) |
| **ANTS\_LIST** | [Ant object] | Holds all ant objects, used to update all ants (default: []) |
| **SPECIES\_LIST** | [Species object] | Holds all species objects, use to update all species (default: []) |
| **RUNNING** | boolean | Determines whether the simulation is running or not (default: false) |
| **GRID\_COLOUR** | string (constant) | The colour of the maps grid lines (default: ‘#000000’) |
| **GRID\_LINE\_WIDTH** | number (constant) | The width of the grid lines (default: 0.2) |
| **BACKGROUND\_COLOUR** | string (constant) | The canvas background colour (default: ‘#C2ABBA’) |
| **OUT\_OF\_BOUNDS\_COLOUR** | string (constant) | The colour of any area off map in the simulation (default: ‘#FFFFFF’) |
| **CELL\_SIZE** | width: integer, height: integer | Size in pixels of a single cell on the map (default: width: 6, height: 6) |
| **CANVAS** | name: string, width: integer,  height: integer | Defines characteristics about the simulations canvas (default: name: ‘simulation’, width: 500, height: 500) |
| **AVERAGE\_FOOD\_SAMPLE\_RATE** | integer (constant) | The number of ticks to wait between sampling for averages of food (default: 10) |
| **NUMBER\_OF\_FIXED\_PLACES** | integer (constant) | The number of numbers after the decimal place to display when displaying values on the interface (default: 2) |
| **SELECTED\_SPECIES** | Species object | Holds the species whose characteristics can be changed in the characteristics panel |
| **SELECTED\_COLOUR** | string (constant) | The colour of the selected species in the data panel (default: ‘#A5C7D9’) |
| **UNSELECTED\_COLOUR** | string (constant) | The default colour for unselected species in the data panel (default: ‘#FFFFFF’) |
| **BUTTON\_UPDATE\_COLOUR** | string (constant) | The colour of the update button when the characteristics of the selected species have been changed by the user (‘#FF0000’) |
| **BUTTON\_NO\_UPDATE\_COLOUR** | string (constant) | The normal colour of a button when no settings have been altered (default: ‘#000000’) |
| **LEFT\_ARROW\_KEY** | integer (constant) | The char code of the left arrow key (default: 37) |
| **RIGHT\_ARROW\_KEY** | integer (constant) | The char code of the right arrow key (default: 39) |
| **UP\_ARROW\_KEY** | integer (constant) | The char code of the up arrow key (default: 38) |
| **DOWN\_ARROW\_KEY** | integer (constant) | The char code of the down arrow key (default: 40) |
| **PLUS\_KEY** | integer (constant) | The char code of the plus key (default: 107) |
| **MINUS\_KEY** | integer (constant) | The char code of the minus/dash key (default: 109) |
| **SPACE\_BAR\_KEY** | integer (constant) | The char code of the space bar key (default: 32) |
| **R\_KEY** | integer (constant) | The char code of the ‘r’ key (default: 82) |
| **S\_KEY** | integer (constant) | The char code of the ‘s’ key (default: 83) |
| **MIN\_ZOOM** | number (constant) | The minimum zoom amount (default: 1) |
| **MAX\_ZOOM** | number (constant) | The maximum zoom amount (default: 14) |
| **PAN\_AMOUNT** | integer (constant) | The number of pixels to pan by for each key press of an arrow key (default: 15) |
| **ZOOM\_AMOUNT** | number (constant) | The amount to zoom for each key press of the plus or minus key (direction depends on key press i.e. plus is zoom in while minus is zoom out) (default: 0.5) |
| **CANVAS\_OFFSET** | x: integer, y: integer | The amount of pixels the simulation on the canvas is offset by from the top left corner of the simulation (default: x: 0, y: 0) |
| **INPUT\_TYPE** | object literal (constant) | Used for creating custom inputs i.e. sliders or buttons |
| **MAP** | [objects] | Holds all objects displayed on the map (default: []) |
| **GRID\_SIZE** | width: integer, height: integer | Size in number of cells of the map (default: width: 250, height: 250) |
| **MAP\_BOUNDARY** | x: {min: integer, max: integer}, y: {min: integer, max: integer} | Used to define the edge of the map i.e. when to wrap the coordinate of the ant e.g. if ant exceeds x.max place the ant on the other side of the map as if the map was a torus |
| **NUM\_OF\_CELLS** | integer | The total number of cells in the map |
| **FOOD\_COLOUR** | string (constant) | The colour of food in the simulation (default: ‘#00FF00’) |
| **FOOD\_HEALTH\_RATIO** | number (constant) | food : health i.e. if = 50 then 1 piece of food is worth 50 health (default: 50) |
| **FOOD\_CHANCE** | number (constant) | The probability of a food source in a cell (default: 0.0004) |
| **STARTING\_QUEEN\_ANT\_NUMBER** | integer | The number of queen ants at the start of the simulation (default: 1) |
| **ANT\_FOOD\_DROP\_SPEED** | integer (constant) | The number of ticks it takes for an ant to drop a piece of food (default: 15) |
| **ANT\_FOOD\_TAKE\_SPEED** | integer (constant) | The number of ticks it takes for an ant to pick up a single piece of food (default: 30) |
| **DAMAGE\_MULTIPLIER** | number (constant) | The amount of health a soldier ant takes does to ants its attacking i.e. if = 100 then 100 \* jawStrength of soldier is the amount of health the ant being attacked would lose each tick it is attacked (default: 100) |
| **NEST\_GUARD\_RADIUS** | integer (constant) | The number of steps ants take once seeing the nest i.e. the sentry radius of solider ants about the nest (default: 100) |
| **SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE** | integer (constant) | The maximum distance between an ant and its target before losing interest (default: 25) |
| **TURN\_RATE** | number (constant) | The rate at which ants turn when guarding i.e. 0.02 means the ant turns 0.02 radians each tick (default: 0.02) |
| **GOAL** | object literal (constant) | Ants goals used to determine what actions the ants need to perform |
| **NEST\_SIZE** | width: integer, height: integer (constant) | The extra size of a nest in cells e.g. {width: 1, height: 1} means go 1 extra cell in both sides and 1 extra cell both up and down (default: width: 1, height: 1) |
| **ANT\_TYPE** | object literal (constant) | Used to show the type of ant |
| **PHEROMONE\_EVAPERATION\_RATE** | number (constant) | The rate at which pheromones in the simulate lose concentration per tick (default: 0.005) |
| **MAX\_PHEROMONE\_CONVENTRATION** | number (constant) | The maximum concentration a pheromone can have (default: 1) |
| **USER\_SPECIES** | Species object | The species used on the first run of the simulation i.e .the first species |
| **CHARS** | Object literal | Holds properties of all characteristics species:   * min – number – The minimum value a characteristic can be * max – number – The maximum value a characteristic can be * type – integer – The type of value a characteristic is i.e. float or integer (use VALUE\_TYPE to define e.g. VALUE\_TYPE.floatValue) * id – string – The HTML ID property of the characteristic * neatName – string – The name used in a label in the HTML * desc – string – The description of what the characteristic does * step – number – The minimum change in the characteristics value * healthModifier – number – The cost of the characteristic (= healthModifier \* value) * defaultValue – number – The default value of the characteristic * value – number – The actual value of the HTML input (differs from the value stored in species as this is formatted for displaying i.e. to a certain number of decimal places) * editable – boolean – Determines if a characteristic can be edited (true if it can be edited else false) * inputType – integer – The type of input required e.g. button (Use INPUT\_TYPE to define e.g. INPUT\_TYPE.slider) |

# Difficult to understand code

Link to design section + algorithms + diagrams

This section contains code snippets which may not be clear at first look

*Note*: comments omitted to reduce size

## The addToMap function

The addToMap function is used in multiple classes for example in Ant:

Ant.prototype.addToMap **=** **function**() {

**if** (**this**.alive) MAP[getCellIndex(**this**.coord)].ant.push(**this**);

};

It adds an ant object to the MAP at the index which the ants coordinate maps to if the ant is still alive. Objects can be added to either MAP[index].ant, MAP[index].food or MAP[index].pheromone depending on the type of object being added.

## The Ant.wonder function

The Ant.wonder function is used by ants when the ant is moving around the map with no specific target e.g. if the ant is searching for something such as when workers are looking for food.

Ant.prototype.wonder **=** **function**() {

**var** M **=** 0;

**var** Mxy **=** {

        x**:** 0,

        y**:** 0

    };

**var** CoM **=** {

        x**:** 0,

        y**:** 0

    };

**var** pheromones **=** **false**;

**for** (**var** i **=** 0; i **<** **this**.pheromonesInRange.length; i**++**) {

**if** (**this**.pheromonesInRange[i].species **===** **this**.species) {

            M **+=** **this**.pheromonesInRange[i].concentration;

            Mxy.y **+=** **this**.pheromonesInRange[i].coord.y **\*** **this**.pheromonesInRange[i].concentration;

            Mxy.x **+=** **this**.pheromonesInRange[i].coord.x **\*** **this**.pheromonesInRange[i].concentration;

            pheromones **=** **true**;

        }

    }

    CoM **=** {x**:** Mxy.x **/** M, y**:** Mxy.y **/** M};

**if** (Math.random() **<** **this**.species.chars.explorativeInfluence)

**this**.prioritizeDirection **=** randDir();

**if** (pheromones **&&** Math.random() **<** **this**.species.chars.pheromoneInfluence) {

**var** angle **=** angleTo(**this**.coord, CoM);

angle **=** validateDirection(angle);

**if** (angle **>** **this**.direction **-** Math.PI **/** 3 **&&** angle **<** **this**.direction **+** Math.PI **/** 3) {

**this**.direction **=** angle;

**this**.prioritizeDirection **=** **this**.direction;

**this**.followingPheromone **=** **true**;

        } **else** {

**this**.direction **=** **this**.prioritizeDirection;

**this**.followingPheromone **=** **false**;

        }

    } **else** {

**this**.direction **=** **this**.prioritizeDirection;

**this**.followingPheromone **=** **false**;

    }

};

The functions purpose is to determine which direction an ant should move, ants are more likely to move towards pheromones. First, the mean direction of all the pheromones in range is calculated weighted by the pheromones concentration CoM (e.g. closer pheromones with a large concentration will have a large effect on the direction then further away pheromones with a lower concentration).

After CoM is calculated there is a chance that the ant will change their prioritizedDirection to a random value. The prioritizeDirection is the direction the ant wants to move in, it is needed so that when an ant is not being influenced by a direction that the ant will move in straight lines rather than just completely randomly.

The ant will have a chance of following the mean pheromone direction, however only if the mean direction is not more than radians from the current direction, this stops ants turning 120° or more around and following a pheromone from the direction they just came. Finally if none of these conditions are true, the ant will follow the prioritizeDirection.

## The die function

The die function is used in a very similar way in multiple classes throughout the simulation, here is how it is implemented in the Ant class:

Ant.prototype.die = **function**() {

**var** index = ANTS\_LIST.indexOf(**this**);

ANTS\_LIST.splice(index, 1);

**var** index = **this**.species.ants.indexOf(**this**);

**this**.species.ants.splice(index, 1);

**this**.alive = **false**;

};

The point of the function is to stop the simulation displaying an object and also remove all references to an object so that garbage collections will destroy the object. To stop an object from being displayed it must not be added to the map, this is the purpose of **this**.alive = false; and **if** (**!this**.alive) **return** void(0); which will stop the object being re added to the map after removal.

To stop the object updating its references must be removes, all ants are stored in ANTS\_LIST (*Note*: both food and pheromones are stored directly on the map), ants must also be removed from **this**.species.ants so that the species has an accurate number of alive ants.

## Food vs. Health

Food on the map contains different amounts of food, when a worker ant collects food it takes a certain amount of food from a food piece and adds it to **this**.carrying. A single amount of food is worth a certain amount of health, this amount is determined by FOOD\_HEALTH\_RATIO. So if a worker eats one piece of food its health will increase by FOOD\_HEALTH\_RATIO. So food is collected by worker ants from the map and converted into health.

## Nest reproducing health

Species have a specific health cost depending on how favourable a species characteristics are. Ants also required a specific amount of health to be created (this is a characteristic of their species). When a nest creates an ant, the ant will be born with an amount of starting health, this value is the amount of health needed to create the ant - the cost the species.

## The reproduce function

The reproduce function is implemented in the species class. It determines which type of ant to create.

Nest.prototype.reproduce **=** **function**() {

**var** prob **=** Math.random();

**var** chars **=** **this**.species.chars;

*// Normalize probabilities and then sort into ascending order*

**var** sum **=** chars.reproductionQueenProb **+** chars.reproductionSoldierProb **+**

chars.reproductionWorkerProb;

**var** queenProb **=** chars.reproductionQueenProb **/** sum;

**var** soldierProb **=** chars.reproductionSoldierProb **/** sum;

**var** workerProb **=** chars.reproductionWorkerProb **/** sum;

**var** ordered **=** [{

prob**:** queenProb,

type**:** ANT\_TYPE.queen

}, {

prob**:** soldierProb,

type**:** ANT\_TYPE.soldier

}, {

prob**:** workerProb,

type**:** ANT\_TYPE.worker

}].sort(**function**(a, b) {

**return** a.prob **-** b.prob;

}); *// sort min to max*

*// Determine which outcome occurred*

**if** (prob **<** ordered[0].prob **&&** **this**.viable(ordered[0].type))

**this**.createAnt(ordered[0].type);

**else** **if** ((prob **<** ordered[1].prob **+** ordered[0].prob) **&&** *// cumulative probability*

**this**.viable(ordered[1].type))

**this**.createAnt(ordered[1].type);

**else** **if** (prob **<** ordered[2].prob **+** ordered[1].prob **+** ordered[0].prob **&&**

**this**.viable(ordered[2].type))

**this**.createAnt(ordered[2].type);

};

This function is called to decide which type of ant to create, at first the ants probability of each type of ant (worker, queen and soldier) are normalized to a sum of one. The normalized values are then ordered into ascending order. A random number is then compared with the normalized values, to determine which probability the random number lies in (*Note*: the probabilities are cumulative as they are normalized thus are added). At each comparison the viability of the ant type is determined, this determines if the nest can afford to create that type of ant i.e. the nest may not have enough health to create a queen and therefore cannot create one. Once the probability which the random number falls into is found and the ant type is viable, the ant which the probability maps to is created.

## Difference between Nest and NestPiece classes

The Nest is an object which represents an ant’s nest, it contains the amount of food the workers deposit into it. A NestPiece is an object used to represent part of a nest on the MAP. Individual nest pieces can be destroyed by soldier ants however the Nest object will continue to exist until there are no NestPieces left.

## Goals

Ants actions are decided by the goal they have been assigned, goals are results which want to be achieved, they often require multiple tasks to be completed. For example a worker may have the goal getFood, this requires the ant to move towards the food, and over a series of ticks pick up individual pieces of food until it cannot carry any more.

All Goals are located in the GOAL constant set in the configuration file. Not all ants can be assigned all goals. Ants are required to have logic in the doTask and updateGoal functions in order to achieve goals.

## The steps variable

The steps variable is used both in queen ants as well as soldier ants. Steps is used to move an ant a certain amount of steps in the direction it is facing. Its value is often small so that an ant will stay in close to proximity to its original location, this is how it is used in the soldier class. When guarding a nest, if a soldier sees the nest, it will move NEST\_GUARD\_RADIUS steps away, which allows it to stay in proximity to the nest without being in direct line of sight of the nest.

Steps is used in the queen class to find a location for a new nest. A queen ant will move a certain number of steps in a specific direction and once it has moved the number of steps required it will create a nest.

## The guardNest function

This function is only used by soldier ants and contains the logic soldier ants use to guard the nest.

Soldier.prototype.guardNest **=** **function**() {

**if** (**this**.nearNest) { *// If the ant is close to the nest*

**if** (**this**.steps **<=** 0 **&&** **!this**.soldiersInView()) { *// and no soldiers in view*

**this**.moving **=** **false**;

**this**.direction **+=** 0.02; *// slowly turn i.e. observing surroundings*

} **else** **if** (**this**.soldiersInView()) { *// If soldiers in view, keep moving*

**this**.nearNest **=** **false**;

}

} **else** **if** (**this**.seeNest() **&&** **!this**.atNest()) { *// If near the nest, move a*

*// specific number of steps away*

**this**.nearNest **=** **true**;

**this**.steps **=** NEST\_GUARD\_RADIUS;

} **else** { *// Otherwise, keep looking for the nest*

**this**.wonder();

**this**.moving **=** **true**;

}

};

When an ant is given the goal to guard its nest (GOAL.guardNest) the ant will try to first find its nest. The ant will wonder around until it is near the nest. The if statement inside the first if block decides whether to stop moving or not. If the ant is not walking away from the nest (i.e. this.steps <= 0) and there are no other friendly soldier ants in view the ant should stop, as its current position is considered a good sentry position. However if there are soldiers in view, the ant should find a new sentry position. This is because to maximise coverage of nests perimeter it makes sense for ants to not double up on positions i.e. no point having two sentries together as they can both see the same area, it makes more sense to spread out sentries as much as possible i.e. so they cannot see the other sentries.

The **else if** (**this**.seeNest() **&&** **!this**.atNest()) is used to set a guard radius. If the conditions apply i.e. the ant can see the nest however is not standing on top of the nest, it should move a certain distance away from the nest to find a good sentry position.

Finally if this is not true the ant should keep looking for the nets.

## The mutate function

Is implemented in the species class and used to mutate a species to represent random mutation of genes in mutation when genetic information is passed on.

Species.prototype.mutate = **function**() {

**if** (Math.random() <= **this**.mutationRate) {

**var** altChars = clone(**this**.chars);

**var** altCharacteristic = randProperty(altChars);

**var** altValue = **this**.mutateChar(altCharacteristic);

altChars[altCharacteristic] = altValue;

**var** species = **this**.createSpecies(altChars);

**return** species;

} **else** {

**return** **this**;

}

};

The function picks a single random characteristic to mutate, and will mutate that characteristic. The function will then create and return a new species which is the same as the current one however with a single mutation in one characteristic.

## this.sleep variable

The this.sleep variable is used by the Ant class. It is used to stop the ant from performing an action for a number of ticks. The value of this.sleep is the number of ticks to wait. An example of its use is in the Ant.takeFood and Ant.move functions

Ant.prototype.takeFood **=** **function**(food) {

**if** (**this**.isFood(food)) { *// If food*

        food.amount **-=** 1; *// Take a single piece of food*

**this**.sleep **+=** ANT\_FOOD\_TAKE\_SPEED;

**if** (**!this**.isFood(food)) *// If food is all gone remove it from the map*

food.removeFromMap();

**return** 1;

    } **else** {

**return** 0;

    }

};

Ant.prototype.move **=** **function**() {

**if** (**this**.sleep **<=** 0) { *// only move when not waiting for a task to complete*

**this**.coord.x **+=** Math.sin(**this**.direction) **\*** **this**.species.chars.speed;

**this**.coord.y **-=** Math.cos(**this**.direction) **\*** **this**.species.chars.speed;

    }

    boundary(**this**.coord, MAP\_BOUNDARY); *// Make sure if the ant is out of bounds*

};

Here it is used to stop the ant from moving for a certain amount of time for each piece of food it collects. This leads to more realistic looking behaviour as it shows that the ant takes time to collect food. If there was not sleep timer the ant would be able to walk over food and be able to pick it up instantly.

# Configuration

## Simulation

All aspects of the simulation can be easily configured by editing config.js.

### Environment

The simulations environment includes how much food is spawned into the environment and how pheromones change over time. The amount of food is controlled using FOOD\_CHANCE this is the probability of a food source in a cell, increase this for more food. The rate at which food will grow is controlled by FOOD\_GROW\_RATE i.e. this is the probability each tick that food in the map will grow. The amount that it grows by is controlled by FOOD\_GROW\_AMOUNT again this is a probability which each piece of food is tested against to determine if the piece of food will grow or not. What's more, if the maximum or minimum size of food needs to be changed it can be easily changed within core\main.js by editing the FoodSystem objects properties variation.min and variation.max. Pheromones within the system evaporate over time, the amount each pheromone evaporates is controlled by PHEROMONE\_EVAPERATION\_RATE. There is also a maximum limit on pheromones i.e. the maximum concentration a pheromone can be, it can be changed by changing MAX\_PHEROMONE\_CONCENTRATION. The simulation starts with a single queen ant of the default species however this can be changed by altering STARTING\_QUEEN\_ANT\_NUMBER, this is useful as it can speed up the development of species during the start of the simulation.

### Multipliers

There are two currencies within the simulation, health and food. The food can only be picked up by worker ants and is transported to the nest where it is converted into health. Health is a measure of how long an ant will live for, it slowly decrease over time until it reaches 0 and the ant dies. The rate of decrease is controlled independently for different ants (as some ants may be more efficient than others at conserving health) to edit the rate of decrease of health in a nest for example, edit the Nest object changing its healthRate property. The amount of health a single piece of food is worth is controlled by the FOOD\_HEALTH\_RATIO configuration variable. Another similar multiplier is the DAMAGE\_MULTIPLIER which sets how much health a single value in jawSize takes away from an ant being attacked i.e. the amount of health taken = DAMAGE\_MULTIPLIER \* jawSize.

### Global characteristics

These are characteristics of ants which are the same for all ants within the simulation.

* ANT\_FOOD\_TAKE\_SPEED - This is the number of ticks it takes for an ant to pick up a single piece of food. Useful to change if you want to be able to see ants spending a longer time collecting food.
* ANT\_FOOD\_DROP\_SPEED - Similar to ANT\_FOOD\_TAKE\_SPEED however this is the number of ticks it takes for an ant to drop a single piece of food off at the nest. This models how in ant nests there is a specific place where food is collected and it may take some time for an ant to navigate to it.
* NEST\_GUARD\_RADIUS - This is the number of steps a solider ant will take away from its nest when guarding it.
* SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE - This is the maximum distance a target ant can be away from a solider ant chasing it before the solider ant gives up interest of chasing it, and picks a new target.
* NEST\_SIZE.width and NEST\_SIZE.height - The number of extra cells around the nest which the nest occupies.

### Species characteristics

The following is a list of the properties each characteristic has:

* min - The minimum values a characteristic can be.
* max - The maximum values a characteristic can be.
* type - The type of value a characteristic is i.e. a float or integer
* id - The HTML ID property of the characteristic. (Useful to edit if want custom styling for certain characteristics).
* neatName - The name used in a label in the HTML
* desc - Describe what the characteristic does (used in hover text)
* step - The minimum change in the characteristics value
* healthModifier - The cost of the characteristic ( = healthModifier \* value)
* defaultValue - The default value
* value - The actual value of the HTML input (differs from the value stored in species as this is formatted for displaying)
* editable - Determines if a characteristic can be edited by the user
* inputType - The type of input required i.e. button

## Interface

To configure the layout of the webpage, edit the index.html and style.css files. This will allow you to change the colour of the webpage or add custom styles. *Note*: core\controls.js can also be edited to change the layout of the configuration and data panels.

### Colours

The following colours can be edited:

* Simulations background colour - BACKGROUND\_COLOUR
* The Grid colour - GRID\_COLOUR
* The out of bounds colour - OUT\_OF\_BOUNDS\_COLOUR
* Selected species colour - SELECTED\_COLOUR and UNSELECTED\_COLOUR
* The update button colours - BUTTON\_UPDATE\_COLOUR and BUTTON\_NO\_UPDATE\_COLOUR
* The colour of food - FOOD\_COLOUR

### Size

The size of the simulation can also be edited. Simply edit CANVAS.width and CANVAS.height variables to change the size of the simulation (*Note*: this may require some changes to style.css to make it look correct). Both the grid size and the size of individual cells can be changed by editing GRID\_SIZE.width, GRID\_SIZE.height and CELL\_SIZE.width, CELL\_SIZE.height respectfully.

### Controls

The simulation has a number of controls which can be changes. The tick rate is the time it takes for a single loop of the entire program, it can be changed by editing the TICK variable (the time in milliseconds of a single tick). Furthermore, zooming and panning can be updated with MIN\_ZOOM, MAX\_ZOOM, ZOOM\_AMOUNT and PAN\_AMOUNT. All shortcuts e.g. space bar to pause the simulation can be easily changed in the core\main.js file, by altering the window.onkeydown case statement.

# Test driven development

Parts of the system have been built using test driven development principles. This means there are a large number of automatic tests written for the system. These will automatically determine if functions are working as expected. These can be run after altering the application in any way to determine if the alteration has broken any other parts of the system. This is clearly useful to more easily track to stops bugs appearing. It is recommended to re run all tests after every major change to the application, this will reduce the number of bugs and errors in the program by highlighting what breaks early in the development.

## Run a test

Uncomment the test in index.html i.e. remove <!— and --> which prepend and postponed the script tag. When the test runs it will output the results of the test in the developer console in the browser.

When running tests make sure to add:

<script src="tests/equal.js" type="text/javascript"></script>

<script src="tests/test.js" type="text/javascript"></script>

<script src="tests/testCase.js" type="text/javascript"></script>

To the index.html so that tests can run.

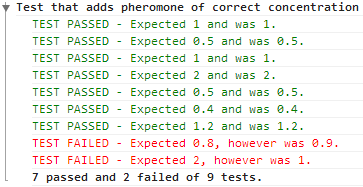
*Note*: An external custom written testing framework is used, to perform tests. As this is not part of the tool and is only used as part of the testing process it is not documented in this document. However its code will be included in the appendix.

### C:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\Testing\Passed distance test.PNGA passed test

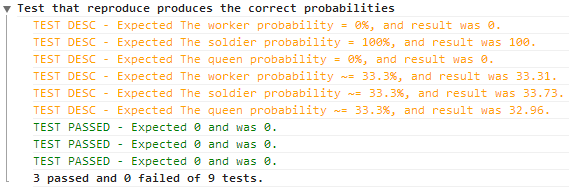
This is an example of the output of a test for the distance function which calculates the distance between two points. For full documentation on a test view the tests source file located in the tests/ directory.

### A failed test

This is an example of the Ant.secrete test failing.



### Human input



Not all tests are completely automatic, some tests require human analysis. Tests which require human analysis are highlighted in orange. This is testing that the Nest.reproduce function produces the correct proportions of the different types of ants depending on a particular set of characteristics. As this is a probabilistic test (due to the randomness used in the function) there is no deterministic outcome. A probability is used over 10,000 runs to determine the probability for each ant. A human is required to evaluate whether the probability is sufficient to accept the test has passed or not.