COSC 343 Assignment 2: Genetic Algorithm Report

Rory Mearns (3928873)

For this assignment a genetic algorithm was implemented to optimise the behaviour of creatures in a simulated world consisting of: the creatures themselves, monsters that eat creatures, food that give the creatures energy and poison that kills the creatures. The setup implemented here closely follows the assignment brief and the behaviours that evolve in the creatures across generations range from the expected to the unusual. The evolution of creature behaviour also proved to be quite sensitive to changes in the way the simulation was setup.

The Setup

In general my simulation followed the specifications set out in the assignment brief very closely.

Language

Although no specific programming language was specified in the brief I deviated from what would could considered a normal choice for this class (Java or Python) and decided to create my implementation using JavaScript. My program runs in a browser and the HTML canvas element was utilised for the visualisation. Results from each generation of the simulation are logged to the console.

World

The world my simulation runs in is a basic grid 60 cells wide and 20 cells high. Any creature, monster, mushroom or strawberry can occupy any one cell with the exception that strawberries and mushrooms cannot occupy the same square.

Below is a list of main variables used to run the simulations discussed in this report:

- World width (cells): 60
- World height (cells): 20
- Time steps per generation: 50
- Generations per simulation: 50
- Number of creatures: 50
- Number of monsters: 15
- Chance of a cell containing a strawberry: 4%
- Chance of a cell containing a mushroom: 4%
- Starting energy of creatures: 100
- Energy gained from eating strawberries: 10
- Chance of a single mutation in a child creature: 1/100

In addition to these I had a single variable to change the wait-time between time steps. To watch the events of a single generation this variable is set to 0.5–1.0 second, to run the simulation quickly and see the results from 50 generations this variable is set to 0.

Initially I ran my simulations for 100 — 200 generations however the most interesting results occurred within the first 50 generations so all subsequent simulations were limited to 50 generations.

Creatures, Monsters and Food

All creatures, monsters and food items were implemented as specified in the assignment brief.

Parents, Offspring and Chromosome Mixing

At the end of each generation creatures were sorted in their array (descending order) based on their energy level (fitness). A normalised fitness value was then assigned to each creature and these were used to calculate and assign an accumulated normalised fitness value for each parent. For each parent selection a new random number between 0 and 1 was generated and the selected parent was the first who's accumulated normalised value was greater than this random number.

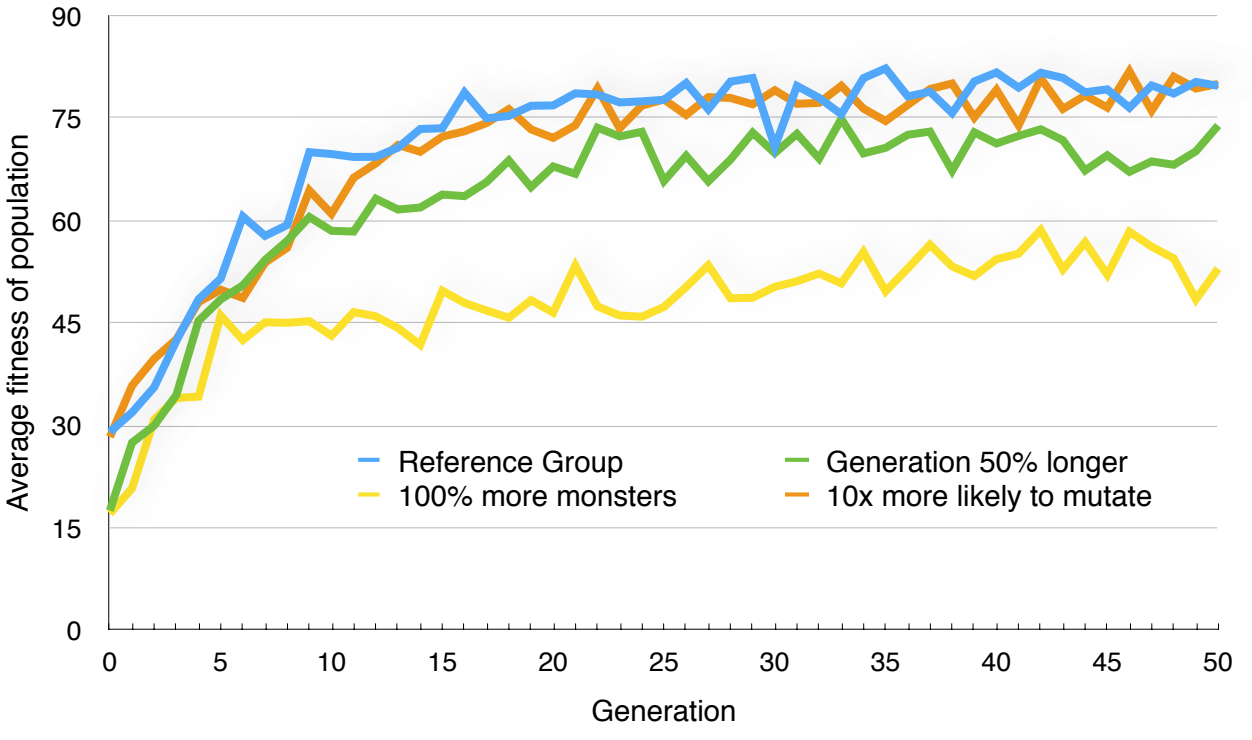
Once two parents are selected their chromosomes are mixed with a chance of random mutation. My algorithm for chromosome mixing ensures that for any given behaviour chromosome taken from one parent the associated weight chromosome for that behaviour must also come from the same parent.

Random mutations are evaluated after all chromosomes have been mixed. Each new child creature has a 1-in-100 chance that one of their chromosomes will be changed to some random new chromosome.

Fitness Change Over Generations

After setting up my simulation as described above I ran the simulation and took the average population fitness for each generation. I did this 10 times then averaged the the results and plotted them (blue) in Figure 1 below. I ran this whole procedure 3 more times, each time making a single change to the the setup relative to the initial reference group.

Figure 1: Average fitness of population at the end of each generation



A fairly rapid increase in the fitness of the population is exhibited over the first 10 generations. This can be explained by the initial rapid spread of 'good' chromosomes and extinction of 'bad' chromosomes. The plateau of the population fitness seen after the first 20 generations may be explained by the number of creatures with 'good' chromosomes competing for a limited food supply beyond this point.

The average fitness of the population at the end of each generation was less when the length of the generations were increased (plotted green in Figure 1) as each creature would have performed more actions over the generation (costing energy and thus reducing individual fitness) and the relative amount of food available would be less.

The average fitness of the population at the end of each generation was even lower when the number of monsters in the world was doubled (plotted yellow in Figure 1).

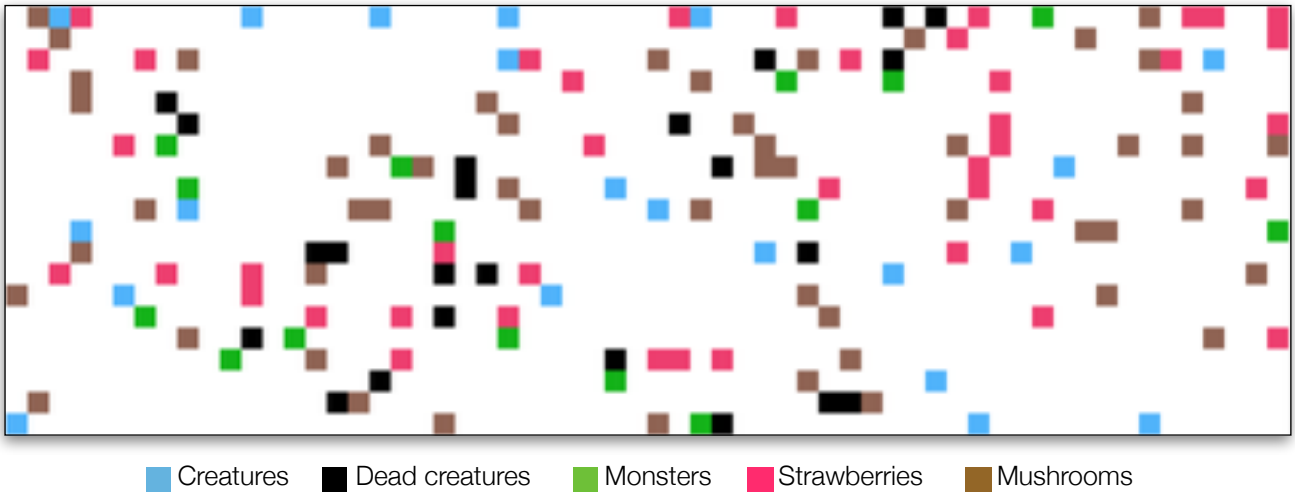
Lastly I ran the simulation with a much higher chance of each child creature undergoing chromosome mutation (plotted orange in Figure 1). I hypothesised that this may help create some 'extremely good' chromosomes and allow the population to break through the plateau in average population fitness seen beyond 20 generations. However this was not the case as the result of a mutation may be positive, negative or neutral. It is likely that these mutations would only benefit the overall population fitness on simulations with significantly more generations (1000+) as a significant number of 'mutation dice rolls' would be needed for very beneficial mutations to occur.

The Effect of Evolution on Behaviour

Figures 2, 3 & 4 below illustrate some of the interesting behaviour changes exhibited by the creatures as they evolved.

Figure 2 shows the state of the simulation at the end of the first generation. The number of black squares illustrate the number of creatures killed either by eating the poisonous mushrooms or by coming in to contact with a monster. The creatures that survived this generation survived by some combination of: avoiding monsters, not eating mushrooms, or luck.

Figure 2: State of the simulation at the end of Generation 1



Mass Migration

Figure 3 shows the state of the simulation at the end of the 50th generation and displays an interesting and unexpected "migrate north" behaviour that was observed on several simulations. In this simulation a chromosome that told the creature to simply 'head north' quickly propagated through the population and was a relatively successful strategy as illustrated by the low number of dead creatures seen in this figure. Any mushroom eating chromosomes were extinct by this state, however it is interesting to note that strawberries were largely ignored by this point too.

Figure 3: State of the simulation at the end of Generation 50

In other simulations when this occurred the migration was always north or south and never east or west. This behaviour is explained by the shape of my world. Because the world is only 20 cells high the most energy any one creature would need to expend to reach the opposite end (north/south) is 20, leaving the creature with at least 80 energy to the end of the generation. Because the fitness of any one creature is a direct measurement of it's energy at the end of a generation these north/south migrating creatures are generally considered to be fit.

When the size of the world is changed to 80*80 this behaviour is sometimes still exhibited but often dies out before the 50th generation as the creature has to traverse a much greater distance to reach the north/south border decreasing its energy level, thus decreasing its fitness and likelihood to be chosen as a parent for subsequent generations.

Mass Extinction

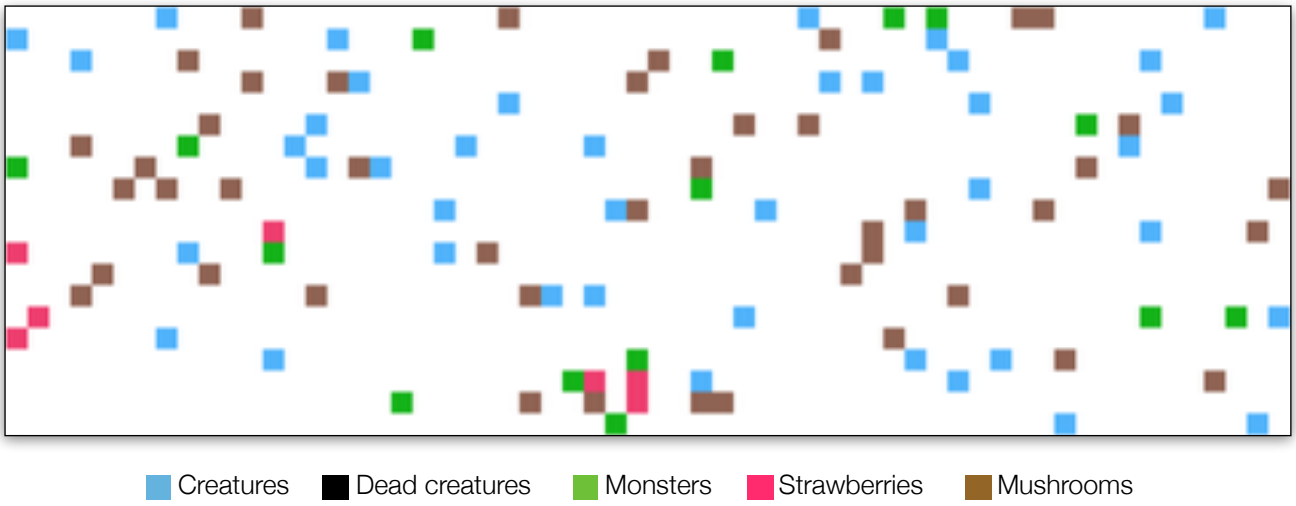
Even more surprising than the mass migration chromosome is that this behaviour sometimes lead to mass extinctions. The mass migration chromosome often consists only of the instruction to migrate north/south and not necessarily to avoid monsters. On several simulations a mass migration was observed, then when one or two monsters happen to approach the border where all of the creatures are congregated they see one of the creatures (their sight is a distance of 2 cells in any direction) and head for that creature. Once they kill that creature they then see the next one from there and a chain reaction occurs killing off a large portion of the population along with the seemingly 'well performing' chromosome all within a single generation.

It is very likely that if the monsters had a 'sight' distance greater than 2 blocks (maybe 4 or even 5 blocks) this migrate north/south chromosome would be less pervasive.

Hungry Creatures

The other behaviour exhibited at the end of the 50th generation is the less surprising but possibly better performing "hunt for food and avoid monsters" behaviour shown in Figure 4. As illustrated in this figure no creatures had been killed by monsters or mushrooms and the chromosome to seek out and then eat strawberries was clearly present in the population shown by the near complete absence of strawberries at the end of this generation. Sometimes this population behaviour evolved out of a migrating population that underwent some mass extinction event.

Figure 4: State of the simulation at the end of Generation 50



Conclusion

Overall the simulation worked well and the evolution of creature behaviour exhibited some interesting and unexpected results. One completely unexpected observation was the rapid spread of a seemingly well performing behaviour (migrate north/south) and then the sudden mass extinction of that behaviour when the conditions in the world become 'just right'.

A natural progression from this point would be to further fine tune and develop the world environment. How would behaviours evolve if food became more scarce from generation to generation? What if opposite edges of the world were connected (i.e. if a creature was at the north border and kept going north it would wrap around and appear at the southern border)? What if monsters needed to eat creatures to survive themselves? What if monsters could squash the strawberries? What if creatures turned into monsters after eating mushrooms? What if elitism was introduced to the offspring creation phase (allowing the top performers to survive subsequent generations)?

All of these developments would undoubtedly have interesting — and possibly unexpected — effects on the evolution of creature behaviour and the world they inhabit.

APPENDIX A: Program Code

genetic.js

```
/* ---- World Variables & Data Structures ---- */
// Timing
var timestep = 0;                // the 'world clock'
var total_frames = 50;           // how many steps in a generation
var generation_clock = 0;        // keep track of the generation
var generations = 50;            // how many generations to run
var wait = 0;                    // how long to wait between timesteps for possible animation

// Drawing
var world_width_cells = 60;             // world width in number of cells
var world_height_cells = 20;            // world height in number of cells
var block_size = 20;                    // how big a 'cell' is in pixels on the screen
var world_width_pixels = world_width_cells*block_size;      // world width in number of pixels
var world_height_pixels = world_height_cells*block_size;    // world height in number of pixels
var creature_color = "#50B3FA";         // color of creatures: blue
var dead_creature_color = "#000000"     // color of dead creatures: black
var monster_color = "#19B319";          // color of monsters: green
var strawberry_color = "#ED3E6F";       // color of strawberries: red
var mushroom_color = "#8F6353";         // color of mushrooms: brown

// Data
var strawb_array = new Array(world_width_cells);     // 2D location array, each cell contains a number indicating
the quantity of food
var mushroom_array = new Array(world_width_cells);   // 2D location array, each cell contains a number indicating
the quantity of food
var num_creatures = 50;                              // number of creatures in the world
var creatures_array = new Array(num_creatures);      // 1D array of all the creatures
var previous_creatures_array = new Array(num_creatures);    // Store the previous generation in
var creatures_location_array = new Array(world_width_cells);// 2D array of all the creatures locations
var num_monsters = 15;                                       // number of monsters in the world
var monsters_array = new Array(num_monsters);                // 1D array of all the monsters
var monsters_location_array = new Array(world_width_cells);// 2D array of all the monsters locations
var chance_of_strawb = 0.04;     // the chance of any one cell containing a strawberry
var chance_of_mush = 0.04;       // the chance of any one cell containing a mushroom
var max_strawb = 6;              // the highest number of food any one strawberry tile can contain
var max_mushroom = 6;            // the highest number of food any one mushroom tile can contain
var energy_from_food = 10;       // how much energy is gained from eating food

// Creature info:
var start_energy = 100;                        // how much energy each creature starts with
var eat_actions = ["eat","ignore"];            // for building the chromosones
var move_actions = ["towards", "away_from", "random", "ignore"];          // for building the chromosones
var default_move_actions = ["random", "north", "east", "south", "west"]; // for building the chromosones

// Canvas
var canvas = document.createElement("canvas");       // Create the Canvas element
var ctx = canvas.getContext("2d");
canvas.width = world_width_pixels;                   // Set the Canvas size based on the size of the world
canvas.height = world_height_pixels;
document.body.appendChild(canvas);


/* ---- Creatures ---- */
function Creature (locationX, locationY) {

      // States & Variables:
      this.locationX = locationX;
      this.locationY = locationY;
      this.energy_level = start_energy;
      this.fitness_value_normalised = undefined;
      this.fitness_value_accumulated = undefined;
      this.actions_list = [];

      // chromosone:
      this.chromosone = new Array(13);
      this.chromosone[0] = eat_actions[Math.floor(Math.random() * 2)];
      this.chromosone[1] = eat_actions[Math.floor(Math.random() * 2)];
      this.chromosone[2] = move_actions[Math.floor(Math.random() * 4)];
      this.chromosone[3] = move_actions[Math.floor(Math.random() * 4)];
      this.chromosone[4] = move_actions[Math.floor(Math.random() * 4)];
      this.chromosone[5] = move_actions[Math.floor(Math.random() * 4)];
      this.chromosone[6] = default_move_actions[Math.floor(Math.random() * 5)];
      this.chromosone[7] = Math.floor(Math.random() * 100) + 1;   // will be an int between 1-100
      this.chromosone[8] = Math.floor(Math.random() * 100) + 1;
      this.chromosone[9] = Math.floor(Math.random() * 100) + 1;
      this.chromosone[10] = Math.floor(Math.random() * 100) + 1;
      this.chromosone[11] = Math.floor(Math.random() * 100) + 1;
      this.chromosone[12] = Math.floor(Math.random() * 100) + 1;
```

```
	// Sensory Functions:
	this.strawb_present = function () {
		if (strawb_array[this.locationX][this.locationY] > 0) {
			return true;
		} else {return false;}
	}

	this.mushroom_present = function () {
		if (mushroom_array[this.locationX][this.locationY] > 0) {
			return true;
		} else {return false;}
	}

	this.nearest_strawb = function () {
		// Check neighborhood:
		// ...first check the squares immediately adjacent:
		for (var i=Math.max(this.locationX-1, 0); i<=Math.min(this.locationX+1, strawb_array.length-1); i++)
{
			for (var j=Math.max(this.locationY-1, 0); j<=Math.min(this.locationY+1,
strawb_array[0].length-1); j++) {
				if (strawb_array[i][j]>0) {

					if (i<this.locationX) {
						return "west";
					}
					else if (i>this.locationX) {
						return "east";
					}
					else if (j<this.locationY) {
						return "north";
					} else {return "south"}
				}
			}
		}
		// ...if there is nothing immediately adjacent, check the next step out:
		for (var i=Math.max(this.locationX-2, 0); i<=Math.min(this.locationX+2, strawb_array.length-1); i++)
{
			for (var j=Math.max(this.locationY-2, 0); j<=Math.min(this.locationY+2,
strawb_array[0].length-1); j++) {
				if (strawb_array[i][j]>0) {

					if (i<this.locationX) {
						return "west";
					}
					else if (i>this.locationX) {
						return "east";
					}
					else if (j<this.locationY) {
						return "north";
					} else {return "south"}
				}
			}
		}
		return false;
	}

	this.nearest_mushroom = function () {
		// Check neighborhood:
		// ...first check the squares immediately adjacent:
		for (var i=Math.max(this.locationX-1, 0); i<=Math.min(this.locationX+1, mushroom_array.length-1); i+
+) {
			for (var j=Math.max(this.locationY-1, 0); j<=Math.min(this.locationY+1,
mushroom_array[0].length-1); j++) {
				if (mushroom_array[i][j]>0) {

					if (i<this.locationX) {
						return "west";
					}
					else if (i>this.locationX) {
						return "east";
					}
					else if (j<this.locationY) {
						return "north";
					} else {return "south"}
				}
			}
		}
		// ...if there is nothing immediately adjacent, check the next step out:
		for (var i=Math.max(this.locationX-2, 0); i<=Math.min(this.locationX+2, mushroom_array.length-1); i+
+) {
			for (var j=Math.max(this.locationY-2, 0); j<=Math.min(this.locationY+2,
mushroom_array.length-1); j++) {
				if (mushroom_array[i][j]>0) {

					if (i<this.locationX) {
						return "west";
					}
					else if (i>this.locationX) {
						return "east";
					}
```

```
                                    else if (j<this.locationY) {
                                          return "north";
                                    } else {return "south"}
                              }
                        }
                  }
                  return false;
            }

            this.nearest_monster = function () {
                  // Check neighborhood:
                  // ...first check the squares immediately adjacent:
                  for (var i=Math.max(this.locationX-1, 0); i<=Math.min(this.locationX+1,
monsters_location_array.length-1); i++) {
                        for (var j=Math.max(this.locationY-1, 0); j<=Math.min(this.locationY+1,
monsters_location_array.length-1); j++) {
                              if (monsters_location_array[i][j]>0) {

                                    if (i<this.locationX) {
                                          return "west";
                                    }
                                    else if (i>this.locationX) {
                                          return "east";
                                    }
                                    else if (j<this.locationY) {
                                          return "north";
                                    } else {return "south"}
                              }
                        }
                  }
                  // ...if there is nothing immediately adjacent, check the next step out:
                  for (var i=Math.max(this.locationX-2, 0); i<=Math.min(this.locationX+2,
monsters_location_array.length-1); i++) {
                        for (var j=Math.max(this.locationY-2, 0); j<=Math.min(this.locationY+2,
monsters_location_array.length-1); j++) {
                              if (monsters_location_array[i][j]>0) {

                                    if (i<this.locationX) {
                                          return "west";
                                    }
                                    else if (i>this.locationX) {
                                          return "east";
                                    }
                                    else if (j<this.locationY) {
                                          return "north";
                                    } else {return "south"}
                              }
                        }
                  }
                  return false;
            }

            this.nearest_creature = function () {
                  // Check neighborhood:
                  // ...first check the squares immediately adjacent:
                  for (var i=Math.max(this.locationX-1, 0); i<=Math.min(this.locationX+1,
creatures_location_array.length-1); i++) {
                        for (var j=Math.max(this.locationY-1, 0); j<=Math.min(this.locationY+1,
creatures_location_array.length-1); j++) {
                              if (creatures_location_array[i][j]==1) {

                                    if (i<this.locationX) {
                                          return "west";
                                    }
                                    else if (i>this.locationX) {
                                          return "east";
                                    }
                                    else if (j<this.locationY) {
                                          return "north";
                                    } else {return "south"}
                              }
                        }
                  }
                  // ...if there is nothing immediately adjacent, check the next step out:
                  for (var i=Math.max(this.locationX-2, 0); i<=Math.min(this.locationX+2,
creatures_location_array.length-1); i++) {
                        for (var j=Math.max(this.locationY-2, 0); j<=Math.min(this.locationY+2,
creatures_location_array.length-1); j++) {
                              if (creatures_location_array[i][j]==1) {

                                    if (i<this.locationX) {
                                          return "west";
                                    }
                                    else if (i>this.locationX) {
                                          return "east";
                                    }
                                    else if (j<this.locationY) {
                                          return "north";
                                    } else {return "south"}
                              }
                        }
```

```
		}
		return false;
	}

	// Actions:
	this.move = function (direction) {
		var dir = direction;

		// If it's random, assign it to a random direction:
		if (dir == "random") {
			dir = default_move_actions[Math.floor(Math.random() * 4) + 1];
		}

		// Do the corresponding for the directions N, E, S, W:
		if (dir == "north" && this.locationY > 0) {
			// Move one block 'south', update: creatures_location_array & this.locationX
			creatures_location_array[this.locationX][this.locationY] = 0;
			creatures_location_array[this.locationX][this.locationY-1] = 1;
			this.locationY--;
			this.energy_level--;
		}
		else if (dir == "east" && this.locationX < world_width_cells-1) {
			// Move one block 'east', update: creatures_location_array & this.locationX
			creatures_location_array[this.locationX][this.locationY] = 0;
			creatures_location_array[this.locationX+1][this.locationY] = 1;
			this.locationX++;
			this.energy_level--;
		}
		else if (dir == "south" && this.locationY < world_height_cells-1) {
			// Move one block 'south', update: creatures_location_array & this.locationX
			creatures_location_array[this.locationX][this.locationY] = 0;
			creatures_location_array[this.locationX][this.locationY+1] = 1;
			this.locationY++;
			this.energy_level--;
		}
		else if (dir == "west" && this.locationX > 0) {
			// Move one block 'west', update: creatures_location_array & this.locationX
			creatures_location_array[this.locationX][this.locationY] = 0;
			creatures_location_array[this.locationX-1][this.locationY] = 1;
			this.locationX--;
			this.energy_level--;
		}
	}

	this.eat = function (food_type) {
		if (food_type == "strawberry") {
			strawb_array[this.locationX][this.locationY]--;
			this.energy_level += energy_from_food;
		}
		else if (food_type == "mushroom") {
			mushroom_array[this.locationX][this.locationY]--;
			this.energy_level = 0;
			creatures_location_array[this.locationX][this.locationY] = 2;
		}
	}

	this.select_action = function () {

		if (this.energy_level == 0) {return;}

		this.actions_list = [];

		// Go through all the senses and add any appropriate actions:
		if (this.strawb_present()!= false && this.chromosone[0] != "ignore") {
			this.actions_list.push(["eat_actions", this.chromosone[0], this.chromosone[7], "strawberry"]);
		}
		if (this.mushroom_present() != false && this.chromosone[1] != "ignore") {
			this.actions_list.push(["eat_actions", this.chromosone[1], this.chromosone[8], "mushroom"]);
		}
		if (this.nearest_strawb() != false && this.chromosone[2] != "ignore") {
			this.actions_list.push(["move_actions", this.chromosone[2], this.chromosone[9],
this.nearest_strawb()]);
		}
		if (this.nearest_mushroom() != false && this.chromosone[3] != "ignore") {
			this.actions_list.push(["move_actions", this.chromosone[3], this.chromosone[10],
this.nearest_mushroom()]);
		}
		if (this.nearest_monster() != false && this.chromosone[4] != "ignore") {
			this.actions_list.push(["move_actions", this.chromosone[4], this.chromosone[11],
this.nearest_monster()]);
		}
		if (this.nearest_creature() != false && this.chromosone[5] != "ignore") {
			this.actions_list.push(["move_actions", this.chromosone[5], this.chromosone[12],
this.nearest_creature()]);
		}

		// If nothing is added to the actions list, do the default:
		if (this.actions_list.length == 0) {
			this.move(this.chromosone[6]);
		} else {
			var action = this.actions_list[0];
```

```
                        var current_weight = this.actions_list[0][2];

                        for (var i=1; i<this.actions_list.length; i++) {
                              if (current_weight<this.actions_list[i][2]) {
                                    action = this.actions_list[i];
                                    current_weight = this.actions_list[i][2];
                              }
                        }
                        // Now do the action:
                        if (action[0] == "eat_actions") {
                              this.eat(action[3]);
                        }
                        else if (action[0] == "move_actions") {
                              if (action[1] == "random") {
                                    this.move("random");
                              }
                              else if (action[1] == "towards") {
                                    this.move(action[3]);
                              }
                              else if (action[1] == "away_from") {
                                    switch (action[3]) {
                                          case "north":
                                          this.move("south");
                                          break;
                                          case "east":
                                          this.move("west");
                                          break;
                                          case "south":
                                          this.move("north");
                                          break;
                                          case "west":
                                          this.move("east");
                                          break;
                                    }
                              }
                        }
                  }
            }
      }

/* ---- Monsters ---- */
function Monster (locationX, locationY) {

      // States:
      this.locationX = locationX;
      this.locationY = locationY;

      // Sensory Functions:
      this.nearest_creature = function () {
            // Check neighborhood:
            // ...first check the squares immediately adjacent:
            for (var i=Math.max(this.locationX-1, 0); i<=Math.min(this.locationX+1,
creatures_location_array.length-1); i++) {
                  for (var j=Math.max(this.locationY-1, 0); j<=Math.min(this.locationY+1,
creatures_location_array.length-1); j++) {
                        if (creatures_location_array[i][j]==1) {

                              if (i<this.locationX) {
                                    return "west";
                              }
                              else if (i>this.locationX) {
                                    return "east";
                              }
                              else if (j<this.locationY) {
                                    return "north";
                              } else {return "south"}
                        }
                  }
            }
            // ...if there is nothing immediately adjacent, check the next step out:
            for (var i=Math.max(this.locationX-2, 0); i<=Math.min(this.locationX+2,
creatures_location_array.length-1); i++) {
                  for (var j=Math.max(this.locationY-2, 0); j<=Math.min(this.locationY+2,
creatures_location_array.length-1); j++) {
                        if (creatures_location_array[i][j]==1) {

                              if (i<this.locationX) {
                                    return "west";
                              }
                              else if (i>this.locationX) {
                                    return "east";
                              }
                              else if (j<this.locationY) {
                                    return "north";
                              } else {return "south"}
                        }
                  }
            }
            return false;
      }
```

```
	// Actions:
	this.move = function (direction) {
		var dir = direction;

		// If it's random, assign it to a random direction:
		if (dir == "random") {
			dir = default_move_actions[Math.floor(Math.random() * 4) + 1];
		}

		// Do the corresponding for the directions N, E, S, W:
		if (dir == "north" && this.locationY > 0) {
			// Move one block 'south', update: monsters_location_array & this.locationX
			monsters_location_array[this.locationX][this.locationY] = 0;
			monsters_location_array[this.locationX][this.locationY-1] = 1;
			this.locationY--;
		}
		else if (dir == "east" && this.locationX < world_width_cells-1) {
			// Move one block 'east', update: monsters_location_array & this.locationX
			monsters_location_array[this.locationX][this.locationY] = 0;
			monsters_location_array[this.locationX+1][this.locationY] = 1;
			this.locationX++;
		}
		else if (dir == "south" && this.locationY < world_height_cells-1) {
			// Move one block 'south', update: monsters_location_array & this.locationX
			monsters_location_array[this.locationX][this.locationY] = 0;
			monsters_location_array[this.locationX][this.locationY+1] = 1;
			this.locationY++;
		}
		else if (dir == "west" && this.locationX > 0) {
			// Move one block 'west', update: monsters_location_array & this.locationX
			monsters_location_array[this.locationX][this.locationY] = 0;
			monsters_location_array[this.locationX-1][this.locationY] = 1;
			this.locationX--;
		}
		collision_check(this.locationX,this.locationY);
	}

	this.select_action = function () {
		// Check for any nearby creatures:
		if (this.nearest_creature() != false) {
			this.move(this.nearest_creature());
		} else {
			this.move("random");
		}
	}
}

/* ---- Operational Functions ---- */
var collision_check = function (x,y) {

	// If a monster collides with a creature it kills it:
	for (var i=0; i<creatures_array.length; i++)   {
		if (creatures_array[i].locationX == x && creatures_array[i].locationY == y) {
			creatures_array[i].energy_level = 0;
			creatures_location_array[x][y] = 2;
		}
	}
};

var step_creatures = function () {

	// Move every creature
	for (var i=0; i<creatures_array.length; i++) {
		creatures_array[i].select_action();
	}

	// Check for collisions:
	for (var i=0; i<monsters_array.length; i++) {
		collision_check(monsters_array[i].locationX, monsters_array[i].locationY);
	}
};

var step_monsters = function () {

	// Move every monster:
	for (var i=0; i<monsters_array.length; i++) {
		monsters_array[i].select_action();
	}
};

var clone_object = function (source) {

	// needed to clone arrays insted of just copying the reference to them:
    for (i in source) {
        if (typeof source[i] == 'source') {
            this[i] = new clone_object(source[i]);
        }
        else{
            this[i] = source[i];
        }
```

```
    }
};

var assign_fitness = function () {

	// First get the sum:
	var sum = 0;
	for (var i=0; i<creatures_array.length; i++) {
		sum += creatures_array[i].energy_level;
	}

	// Then assign the fitness_value_normalised and calculate the fitness_value_accumulated:
	var accumulation = 0;
	for (var i=0; i<creatures_array.length; i++) {
		creatures_array[i].fitness_value_normalised = creatures_array[i].energy_level/sum;
		accumulation += creatures_array[i].fitness_value_normalised;
		creatures_array[i].fitness_value_accumulated = accumulation;
	}
};

var create_offspring = function () {
	// My implimentation of this first creates a whole new generation of random creatures then
	//     'edits' their chromosomes based on their parents.

	for (var i=0; i<creatures_array.length; i++) {

		// First find two DIFFERENT parents:
		var parent_1 = find_parent();
		var parent_2 = find_parent();
		while (parent_1 == parent_2) {
			parent_2 = find_parent();
		}

		// Then assign the chromosomes from the two different parents:
		// Note if a given 'ACTION' is taken from parent_1, that actions associated
		//     'WEIGHT' is also taken from the same parent:
		creatures_array[i].chromosone[0] = parent_1.chromosone[0];
		creatures_array[i].chromosone[1] = parent_2.chromosone[1];
		creatures_array[i].chromosone[2] = parent_1.chromosone[2];
		creatures_array[i].chromosone[3] = parent_2.chromosone[3];
		creatures_array[i].chromosone[4] = parent_1.chromosone[4];
		creatures_array[i].chromosone[5] = parent_2.chromosone[5];
		creatures_array[i].chromosone[6] = parent_1.chromosone[6];
		creatures_array[i].chromosone[7] = parent_1.chromosone[7];
		creatures_array[i].chromosone[8] = parent_2.chromosone[8];
		creatures_array[i].chromosone[9] = parent_1.chromosone[9];
		creatures_array[i].chromosone[10] = parent_2.chromosone[10];
		creatures_array[i].chromosone[11] = parent_1.chromosone[11];
		creatures_array[i].chromosone[12] = parent_2.chromosone[12];

		// With a chance of 1 in 100:
		if ((Math.floor(Math.random() * 99) + 1) == 5) {

		// Cause a mutation
		var r = Math.floor(Math.random() * 12);
			switch (r) {
				case 0:
				creatures_array[i].chromosone[0] = eat_actions[Math.floor(Math.random() * 2)];
				break;
				case 1:
				creatures_array[i].chromosone[1] = eat_actions[Math.floor(Math.random() * 2)];
				break;
				case 2:
				creatures_array[i].chromosone[2] = move_actions[Math.floor(Math.random() * 4)];
				break;
				case 3:
				creatures_array[i].chromosone[3] = move_actions[Math.floor(Math.random() * 4)];
				break;
				case 4:
				creatures_array[i].chromosone[4] = move_actions[Math.floor(Math.random() * 4)];
				break;
				case 5:
				creatures_array[i].chromosone[5] = move_actions[Math.floor(Math.random() * 4)];
				break;
				case 6:
				creatures_array[i].chromosone[6] = default_move_actions[Math.floor(Math.random() * 5)];
				break;
				case 7:
				creatures_array[i].chromosone[7] = Math.floor(Math.random() * 100) + 1;
				break;
				case 8:
				creatures_array[i].chromosone[8] = Math.floor(Math.random() * 100) + 1;
				break;
				case 9:
				creatures_array[i].chromosone[9] = Math.floor(Math.random() * 100) + 1;
				break;
				case 10:
				creatures_array[i].chromosone[10] = Math.floor(Math.random() * 100) + 1;
				break;
				case 11:
				creatures_array[i].chromosone[11] = Math.floor(Math.random() * 100) + 1;
```

```
                        break;
                        case 12:
                        creatures_array[i].chromosone[12] = Math.floor(Math.random() * 100) + 1;
                        break;
                  }
            }
      }
};

var find_parent = function () {

      var r = Math.random();
      for (var j=0; j<num_creatures; j++) {
            var fit = previous_creatures_array[j].fitness_value_accumulated;
            if (fit>r) {
                  return previous_creatures_array[j];
            }

      }
};


/* ---- Draw Everything ---- */
var render = function () {

      // If something has moved, remove the rectange:
      for (var i=0; i<world_width_cells; i++) {
            for (var j=0; j<world_height_cells; j++) {
                  var x = i*block_size;
                  var y = j*block_size;

                  if (creatures_location_array[i][j] == 0) {
                        ctx.clearRect(x, y, block_size, block_size);
                        creatures_location_array[i][j] = undefined;
                  }
                  if (monsters_location_array[i][j] == 0) {
                        ctx.clearRect(x, y, block_size, block_size);
                        monsters_location_array[i][j] = undefined;
                  }
            }
      }

      // draw the strawberry & mushroom map:
      for (var i=0; i<world_width_cells; i++) {
            for (var j=0; j<world_height_cells; j++) {
                  var x = i*block_size;
                  var y = j*block_size;

                  if (strawb_array[i][j]>0) {
                        ctx.fillStyle = strawberry_color;
                        ctx.fillRect(x, y, block_size, block_size);
                  }
                  if (mushroom_array[i][j]>0) {
                        ctx.fillStyle = mushroom_color;
                        ctx.fillRect(x, y, block_size, block_size);
                  }
            }
      }

      // draw the creatures on the map:
      for (var i=0; i<world_width_cells; i++) {
            for (var j=0; j<world_height_cells; j++) {
                  var x = i*block_size;
                  var y = j*block_size;

                  if (creatures_location_array[i][j] == 1) {
                        ctx.fillStyle = creature_color;
                        ctx.fillRect(x, y, block_size, block_size);
                  }
                  else if (creatures_location_array[i][j] == 2) {
                        ctx.fillStyle = dead_creature_color;
                        ctx.fillRect(x, y, block_size, block_size);
                  }


            }
      }

      // draw the monsters on the map:
      for (var i=0; i<world_width_cells; i++) {
            for (var j=0; j<world_height_cells; j++) {
                  var x = i*block_size;
                  var y = j*block_size;

                  if (monsters_location_array[i][j] == 1) {
                        ctx.fillStyle = monster_color;
                        ctx.fillRect(x, y, block_size, block_size);
                  }
            }
      }
};
```

```
/* ---- Initialise Everything ---- */
var initialise = function () {
	// Clear Arrays for Rendering:
	creatures_location_array = new Array(world_width_cells);
	monsters_location_array = new Array(world_width_cells);
	strawb_array = new Array(world_width_cells);
	mushroom_array = new Array(world_width_cells);

	// Clear canvas:
	ctx.clearRect (0 , 0, world_width_pixels, world_height_pixels);

	// fill strawberry & mushroom array:
	for (var i=0; i<world_width_cells; i++) {
		strawb_array[i] = new Array(world_height_cells);
		mushroom_array[i] = new Array(world_height_cells);

		for (var j=0; j<world_height_cells; j++) {
			var rand = Math.random();

			if (rand<chance_of_strawb) {
				strawb_array[i][j] = Math.floor(Math.random() * max_strawb) + 1;
				mushroom_array[i][j] = 0;
			}
			else if (rand>(1-chance_of_mush)) {
				mushroom_array[i][j] = Math.floor(Math.random() * max_mushroom) + 1;
				strawb_array[i][j] = 0;
			} else {
				mushroom_array[i][j] = 0;
				strawb_array[i][j] = 0;
			}
		}
	}

	// Create sparse array of monster and creature locations:
	for (var i=0; i<world_width_cells; i++) {
		monsters_location_array[i] = new Array(world_height_cells);
		creatures_location_array[i] = new Array(world_height_cells);
	}

	// fill creatures_array
	for (var i=0; i<num_creatures; i++) {
		// find a random location in the 2D array:
		var x = Math.floor(Math.random() * world_width_cells);
		var y = Math.floor(Math.random() * world_height_cells);

		if (creatures_location_array[x][y] == undefined) {
			creatures_array[i] = new Creature(x,y);
			creatures_location_array[x][y] = 1;
		} else {
			i--;
		}
	}

	// fill monsters_array:
	for (var i=0; i<num_monsters; i++) {
		// find a random location in the 2D array:
		var x = Math.floor(Math.random() * world_width_cells);
		var y = Math.floor(Math.random() * world_height_cells);

		if (monsters_location_array[x][y] == undefined){
			monsters_array[i] = new Monster(x,y);
			monsters_location_array[x][y] = 1;
		} else {
			i--;
		}
	}

	// Check for any immediate monster+creature collisions:
	for (var i=0; i<monsters_array.length; i++) {

		for (var j=0; j<creatures_array.length; j++) {

			if (monsters_array[i].locationX == creatures_array[j].locationX && monsters_array[i].locationY
== creatures_array[j].locationY) {
				// console.log("Collision on initialisation! Creature instantly killed");
				creatures_array[j].energy_level = 0;
				creatures_location_array[creatures_array[j].locationX][creatures_array[j].locationY] ==
2;
			}
		}
	}
};


/* ---- Program Funcitons ---- */
var main = function () {

	// Step creatures every time step:
	step_creatures();
```

```
	// Step monsters every second time step:
	if (timestep%2 == 0) {step_monsters();}

	// Re-render everything and increment timestep:
	render();
	timestep++;

	// Loop if we are not at the end of the generation, otherwise sort the creatures_array:
	if (timestep<total_frames) {
		setTimeout(function () {requestAnimationFrame(main);}, wait);
	}
	else if (timestep >= total_frames && generation_clock < generations){ // What to do at the end of each
generation:

		// Sort the creatures array and assign their fitness:
		creatures_array.sort(function(obj1, obj2) {return obj2.energy_level - obj1.energy_level;});
		assign_fitness();
		previous_creatures_array = new clone_object(creatures_array);

		// Print out the average population of the generation:
		var sum = 0;
		for (var i=0; i<creatures_array.length; i++) {
			sum += creatures_array[i].energy_level;
		}
		var average = sum/creatures_array.length;
		console.log("Generation " + generation_clock + "\t" + average);

		// Set up a new world for the next generation:
		initialise();
		create_offspring();
		render();
		generation_clock++;
		timestep = 0;

		// Loop back to the start of main
		setTimeout(function () {requestAnimationFrame(main);}, wait);

	} else {	// If we are at the end of the last generation:

		creatures_array.sort(function(obj1, obj2) {return obj2.energy_level - obj1.energy_level;});

		// Print out the average population of the generation:
		var sum = 0;
		for (var i=0; i<creatures_array.length; i++) {
			sum += creatures_array[i].energy_level;
		}
		var average = sum/creatures_array.length;
		console.log("Generation " + generation_clock + "\t" + average);

		// Log out that we are at the end!
		console.log("Finished");
	}
};


/* ---- Running the Program ---- */
// First initialise everything
initialise();

// Then render everything on the screen
render();

// Then head on in to the main function
main();
```

index.html

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>COSC 343 Assignment 2</title>
	<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
	<h1>COSC 343: Assignment 2</h1>
	<p>Rory Mearns ID.3928873</p>
	<div id="world">
		<script src="genetic.js"></script>
	</div>
	<div id="key">
		<h2>Key:</h2>
		<ul>
			<li><div class="key-item" id="blue"></div>Creatures</li>
			<li><div class="key-item" id="black"></div>Dead Creatures</li>
```

```
                    <li><div class="key-item" id="green"></div>Monsters</li>
                    <li><div class="key-item" id="red"></div>Strawberries</li>
                    <li><div class="key-item" id="brown"></div>Mushrooms</li>
                </ul>
        </div>
</body>
</html>
```

style.css

```
body {
        background-color: #E6E6E6;
        font-family: "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lucida
Grande", sans-serif;
        font-weight: 300;
}

h1 {
        text-align: center;
}

h2 {
        margin-top: 0;
        margin-bottom: 5px;
}

p {
        text-align: center;
}

ul {
        list-style: none;
        margin-top: 0;
        padding-left: 0;
        margin-left: 0;
}

li {
        margin-bottom: 5px;
        padding-left: 0;
        margin-left: 0;
}

canvas {
        border: white 1px solid;
        margin-left: auto;
        margin-right: auto;
        display: block;
        background-color: white;

}

.key-item {
        width: 14px;
        height: 14px;
        margin-right: 8px;
        display: inline-block;
}

#key {
        width: 250px;
        margin: 20px auto 0 auto;
        background-color: white;
        border-radius: 2px;
        padding: 5px 0 5px 20px;
}

#blue {
        background-color: #50B3FA;
}

#green {
        background-color: #19B319;
}

#red {
        background-color: #ED3E6F;
}

#brown {
        background-color: #8F6353;
}

#black {
        background-color: #000000;
}
```