LiveView of Evolution!

Recreating Life With Elixir Processes

Johnny Winn

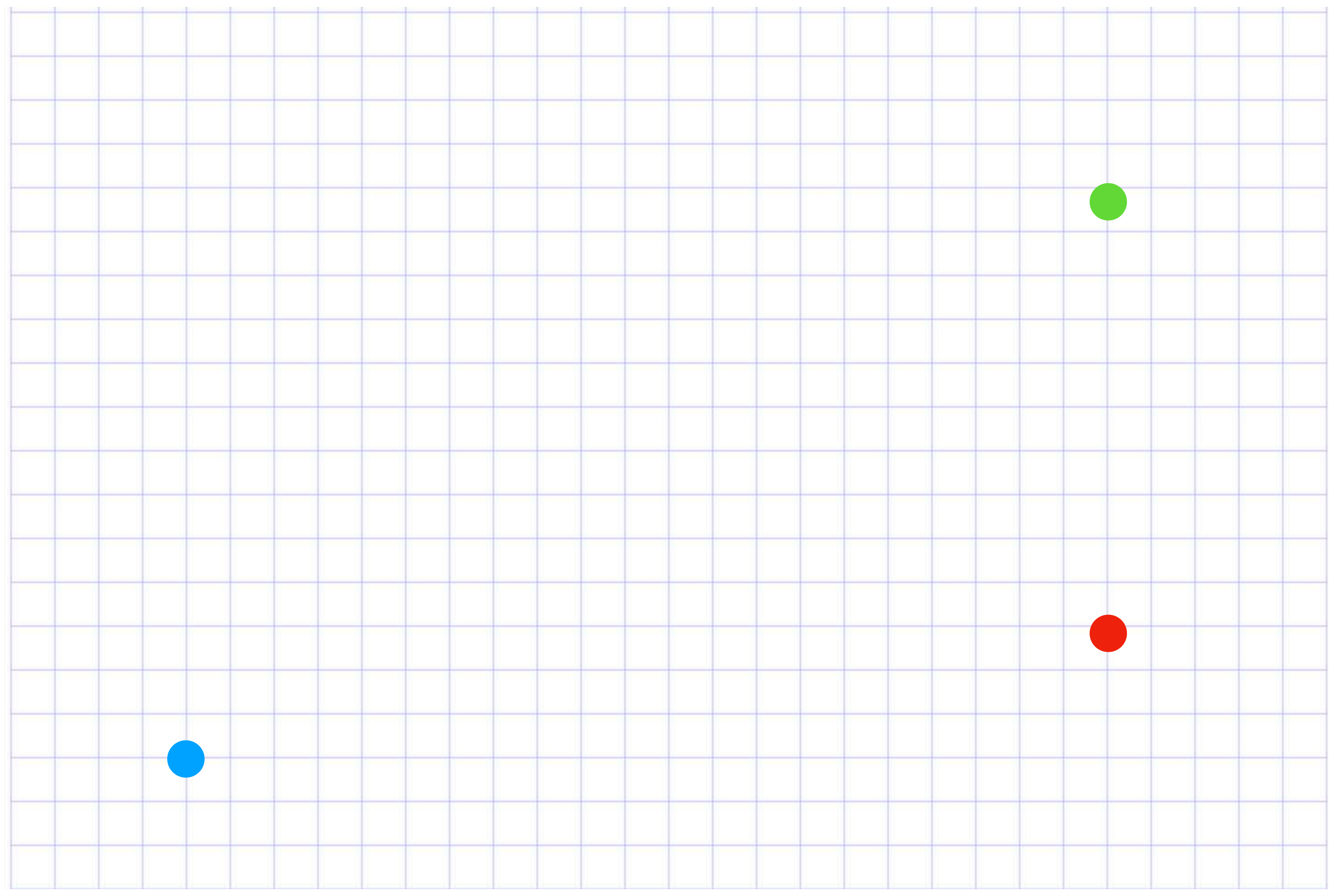
Sr Software Engineer @ Weedmaps

Host of the Elixir Fountain

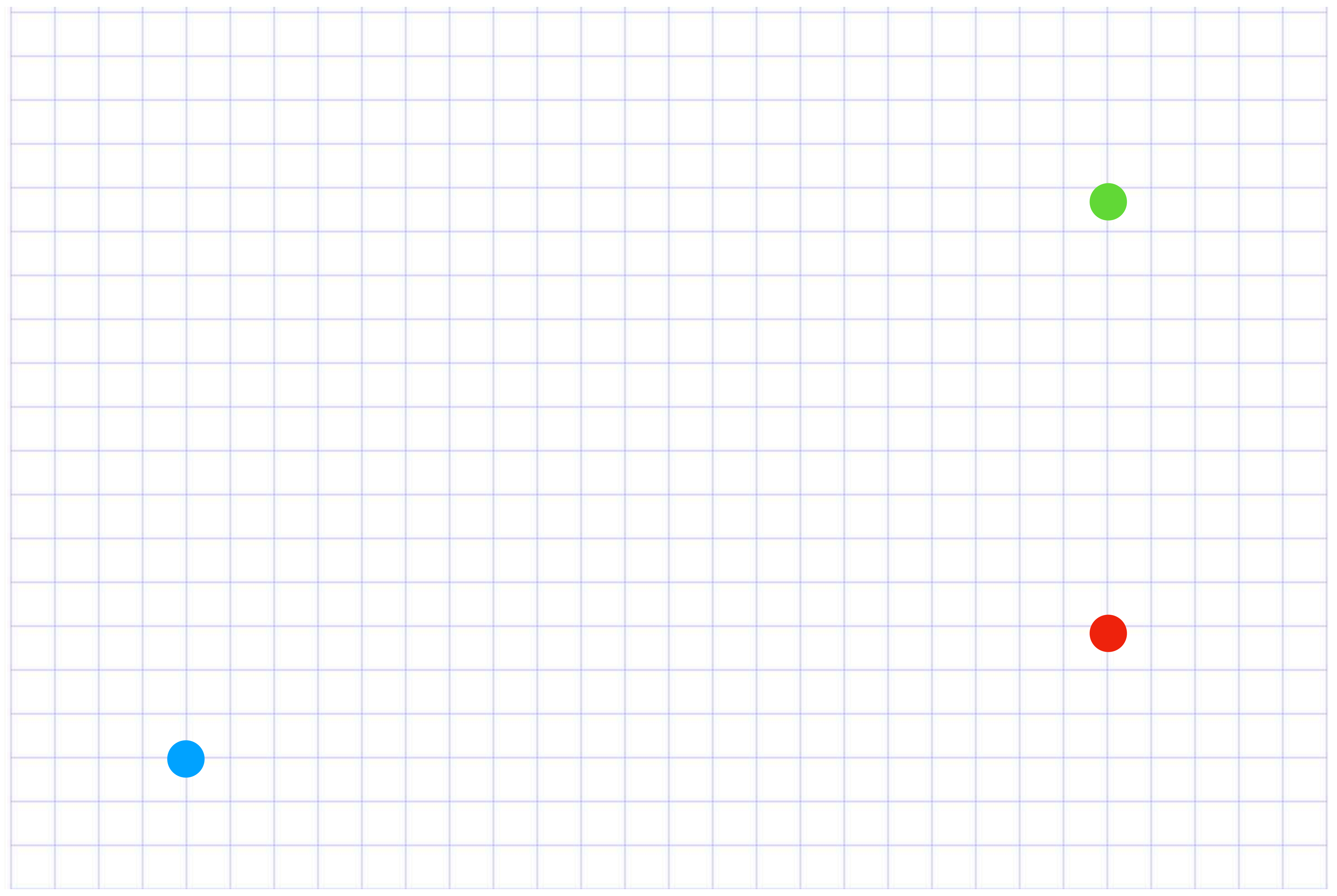
@johnny_rugger

github: nurugger07

https://github.com/nurugger07/prototype



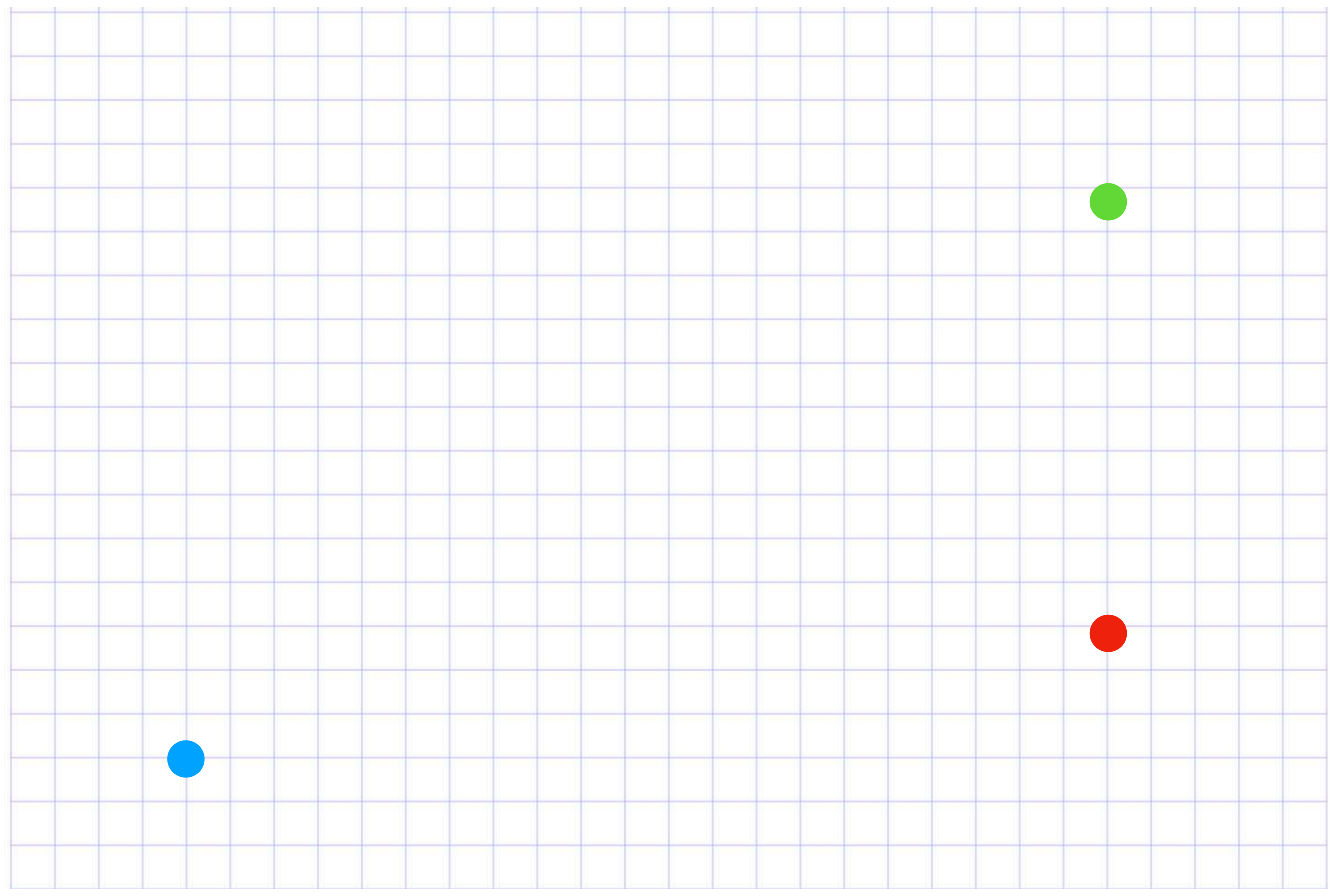
* The Experiment



https://github.com/nurugger07/prototype

* The Experiment

* Making the Rules

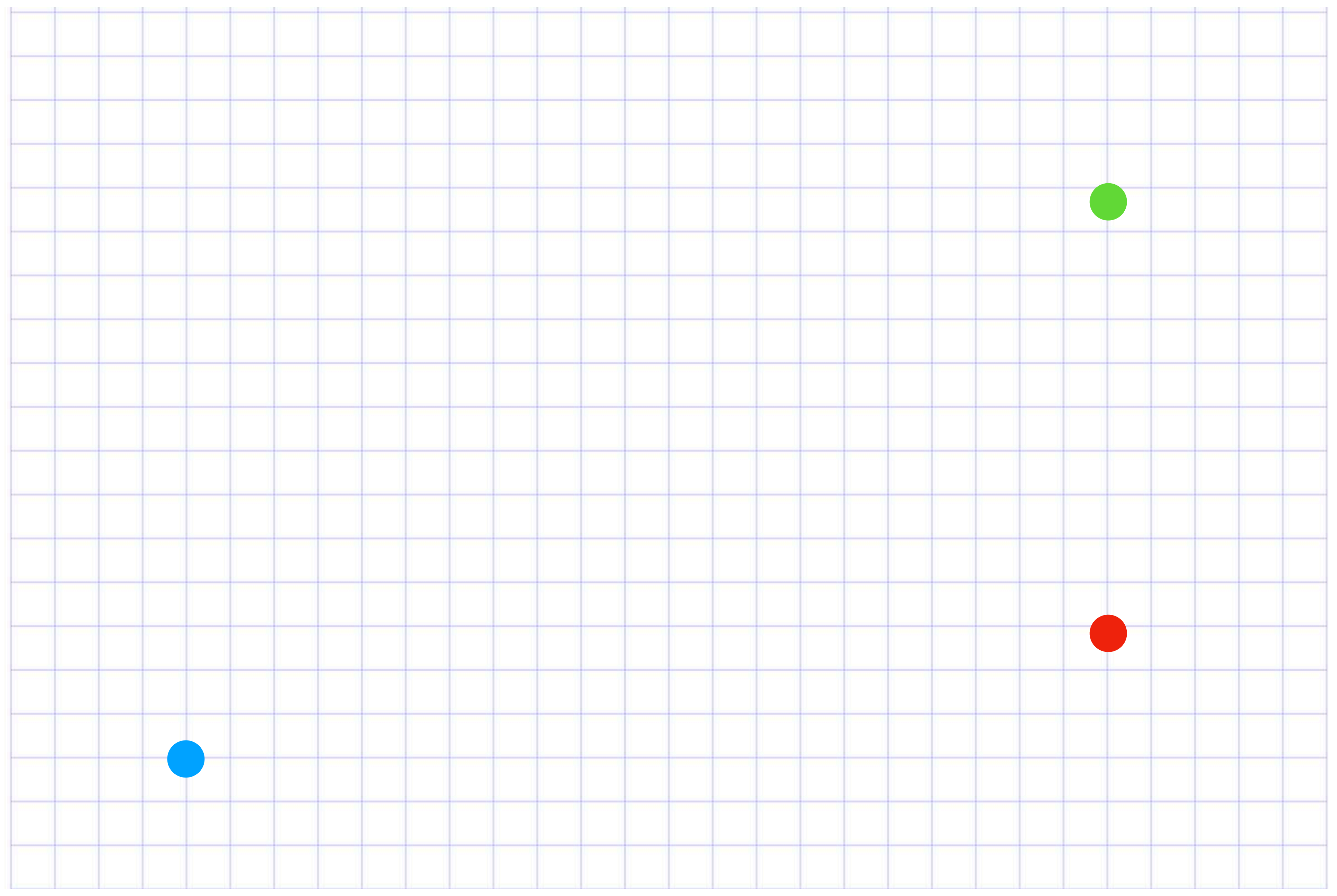


https://github.com/nurugger07/prototype

* The Experiment

* Making the Rules

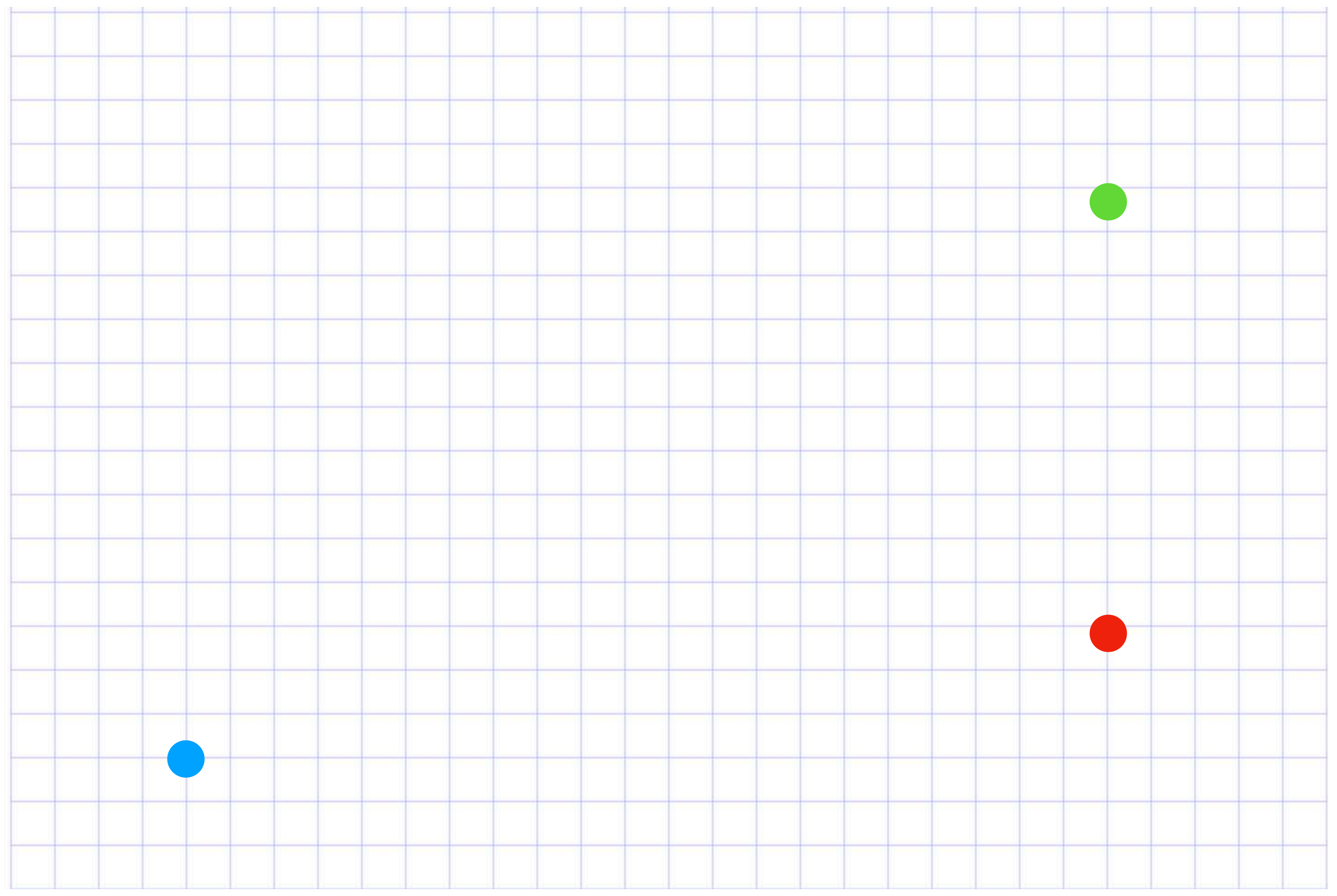
* Building the Playground



https://github.com/nurugger07/prototype

* The Experiment

Why genetics Johnny?



* The Experiment

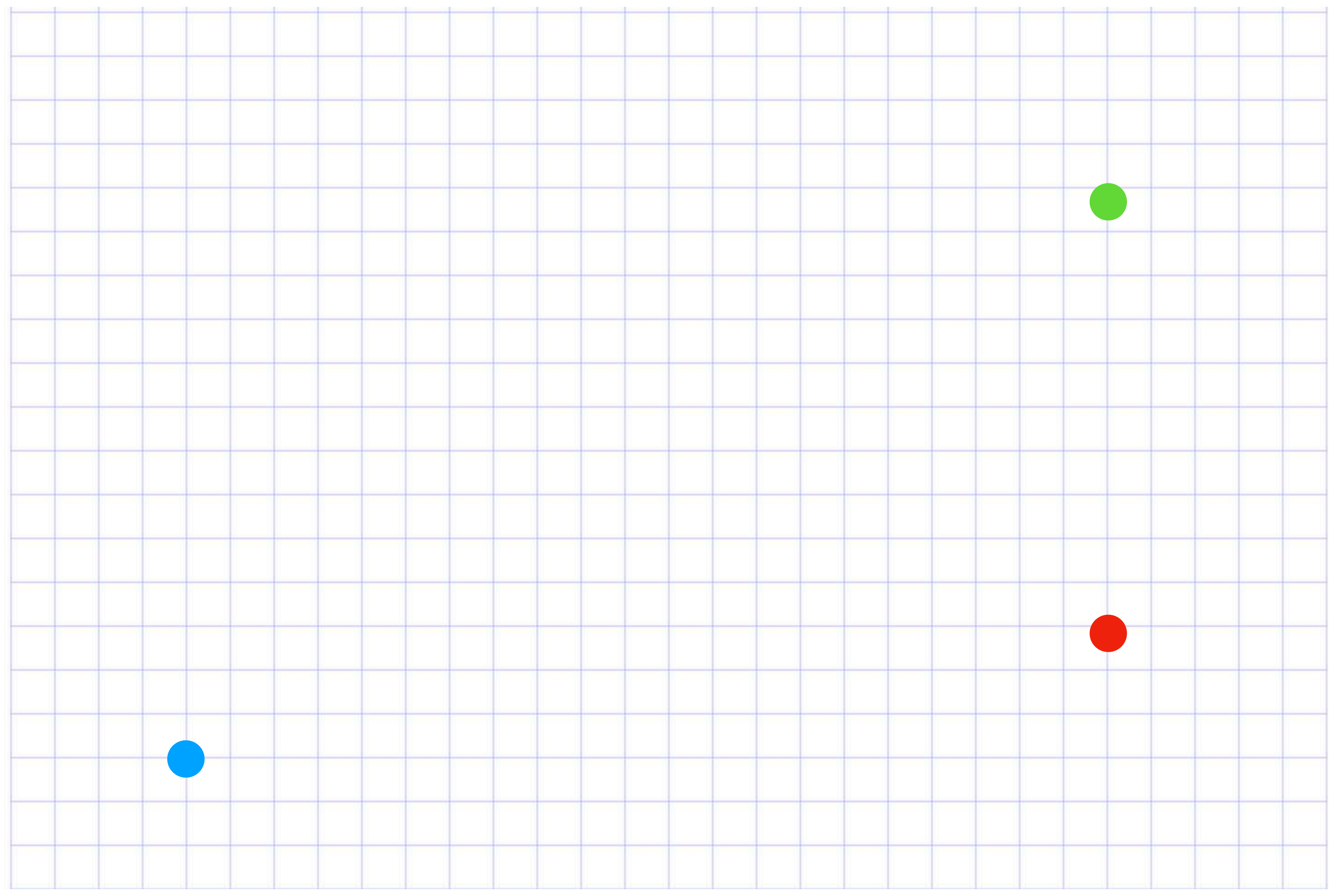
Why genetics Johnny?

* The Experiment

Why genetics Johnny?

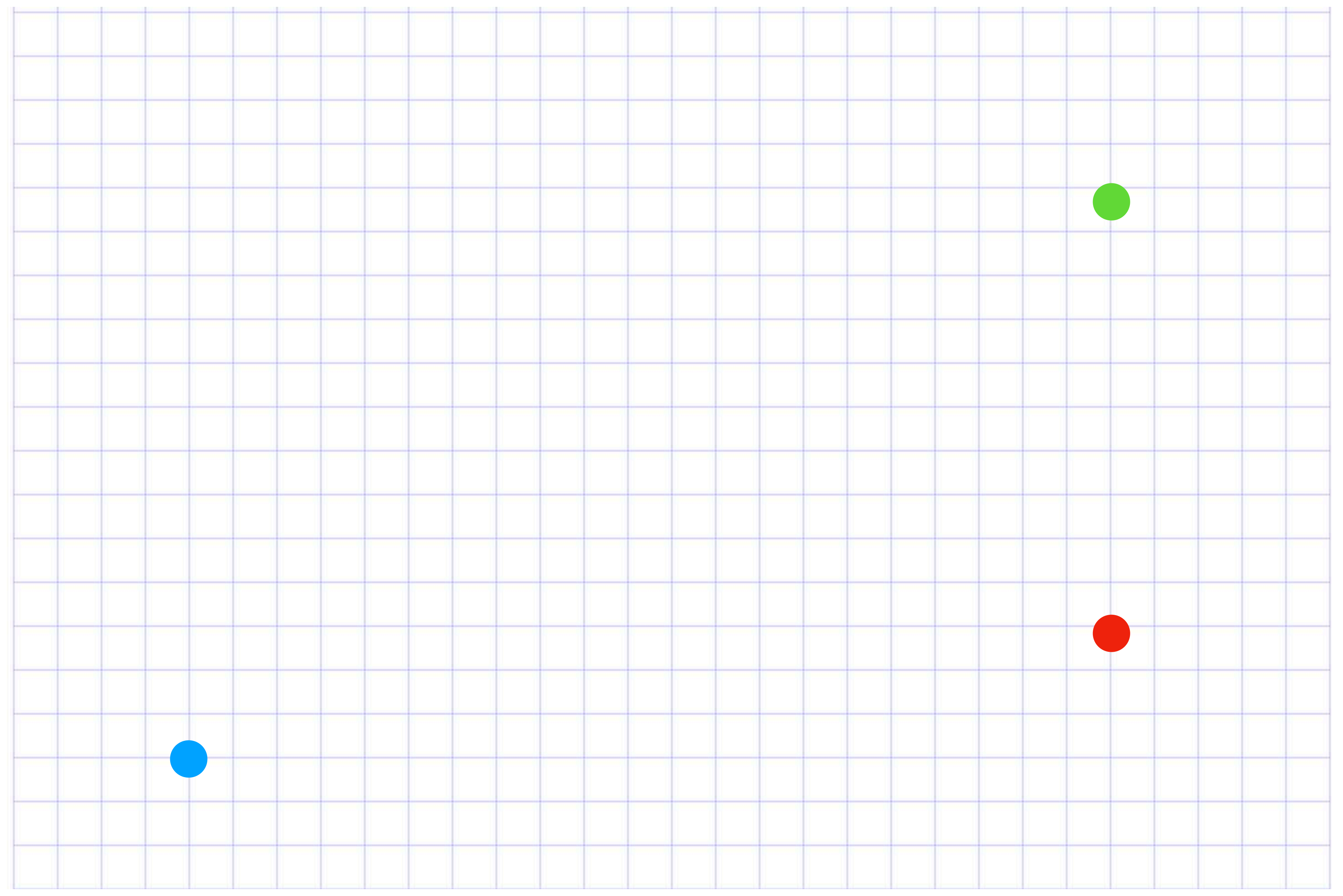
Hello, Little World!

* Nearest Neighbor



* Nearest Neighbor

* Trajectory



* Nearest Neighbor

* Trajectory

* Collision Detection

* Nearest Neighbor

* Trajectory

* Collision Detection

* Fitness

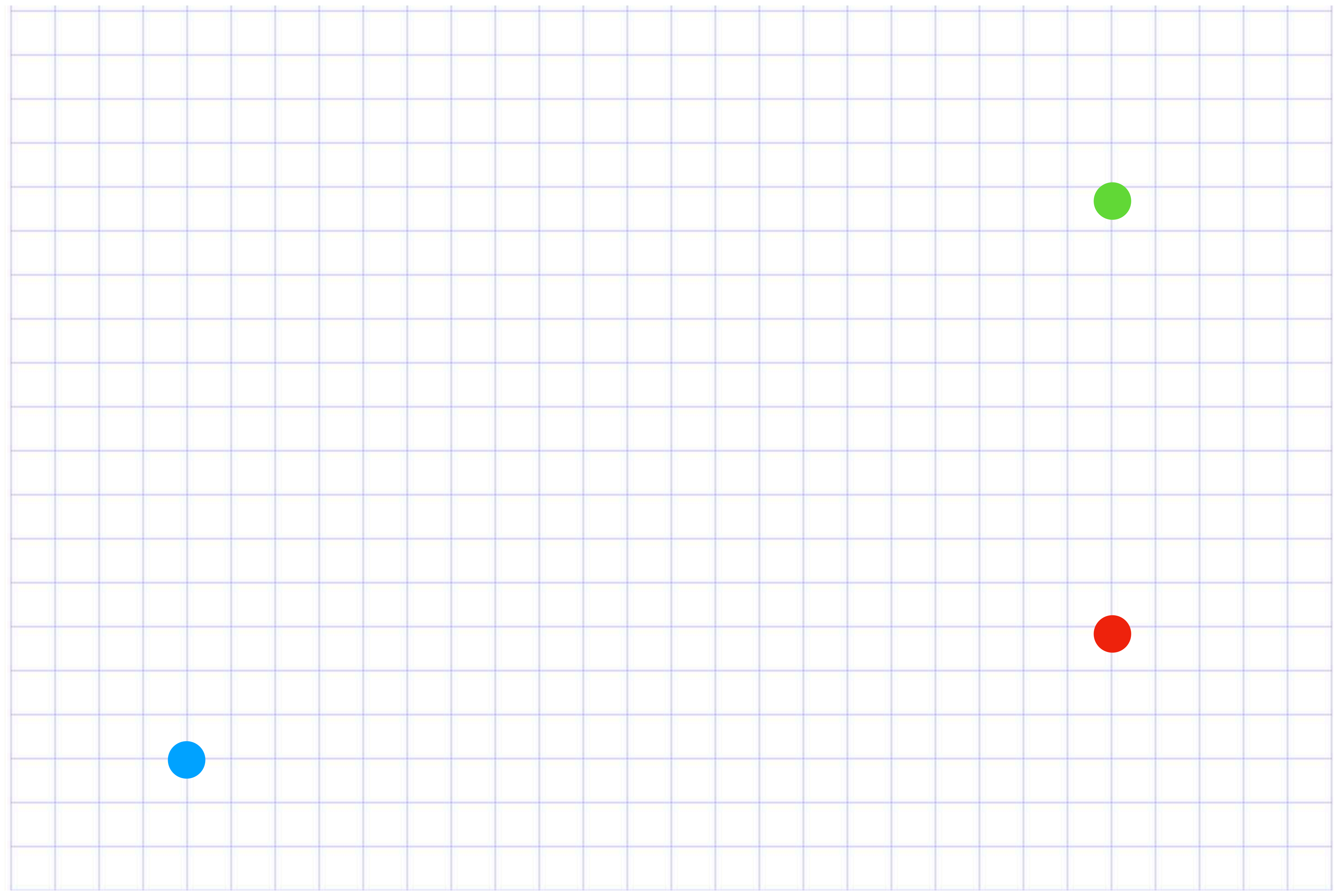
* Nearest Neighbor

* Trajectory

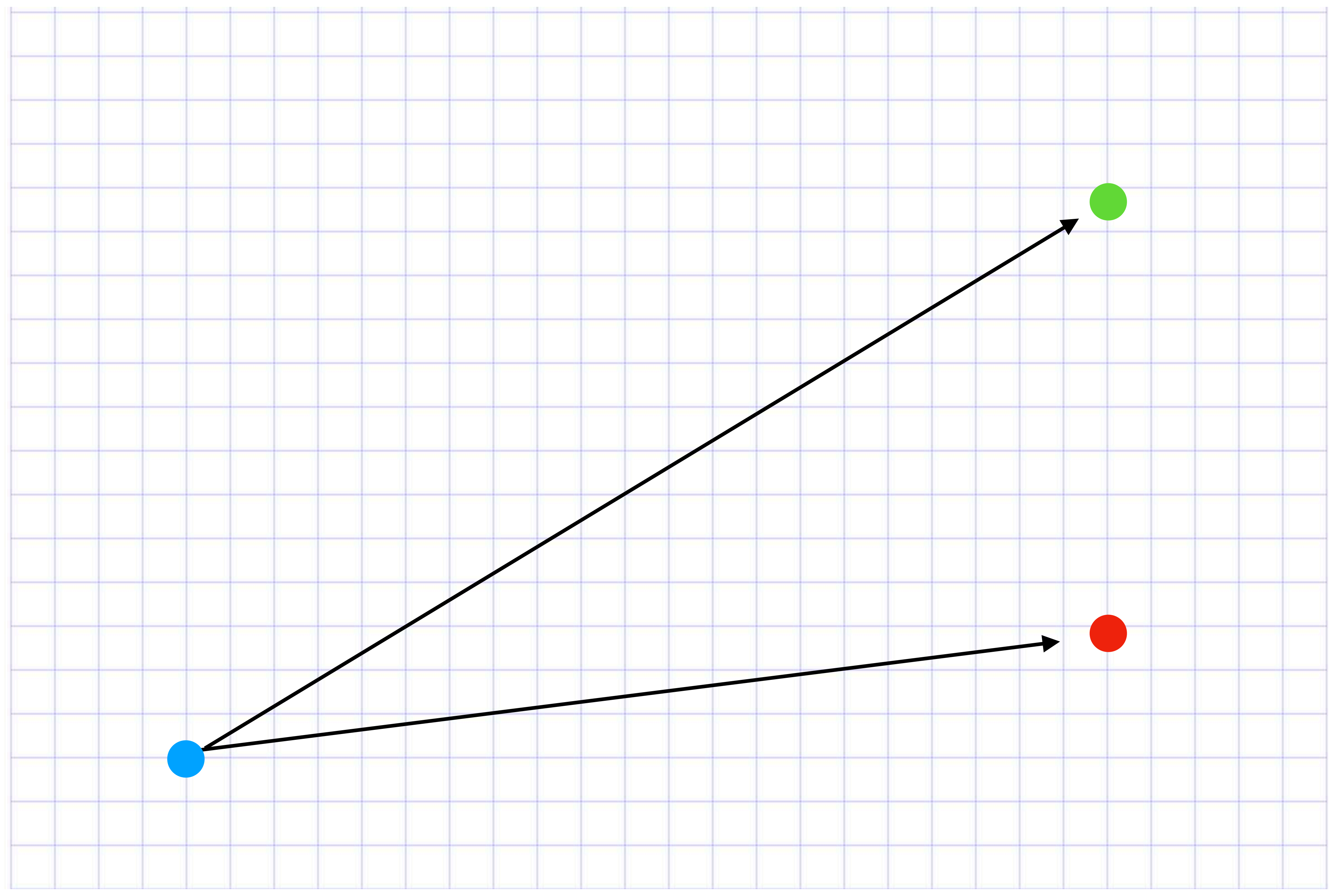
* Collision Detection

* Fitness

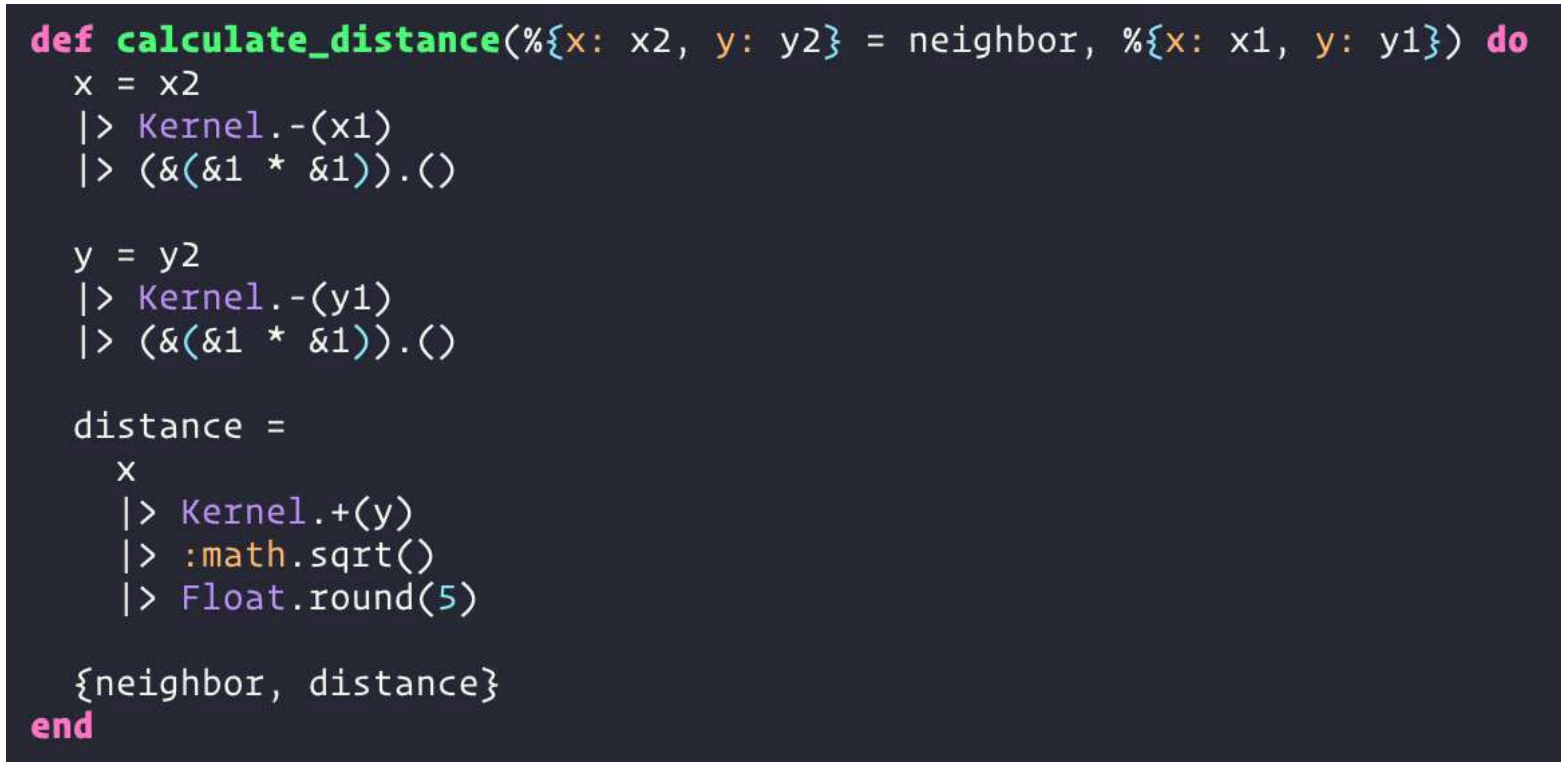
* Assigning Traits



* Nearest Neighbor

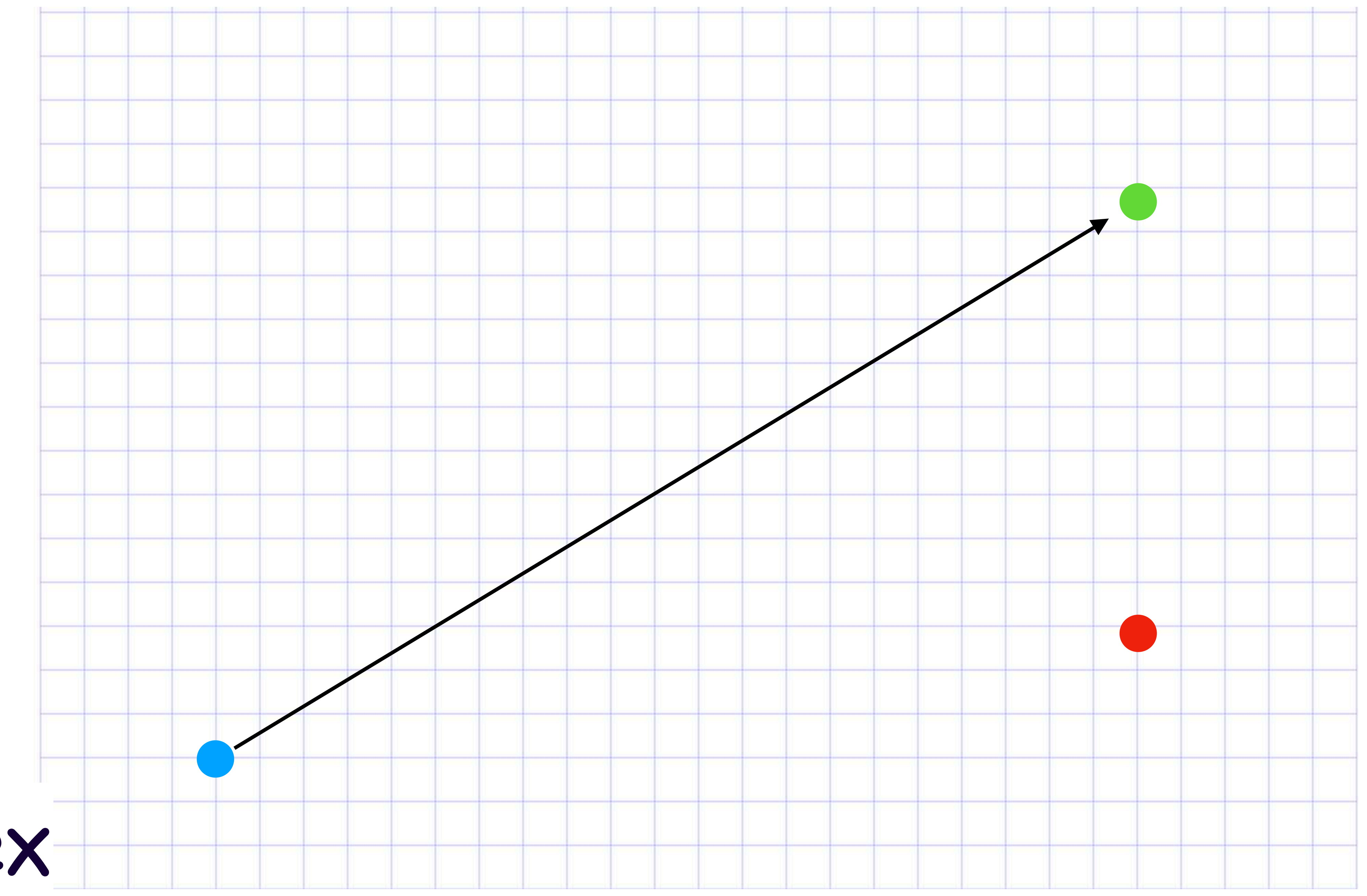


* Nearest Neighbor



```elixir
def calculate_distance(%{x: x2, y: y2} = neighbor, %{x: x1, y: y1}) do
  x = x2
  |> Kernel.-(x1)
  |> (&(&1 * &1)).()

  y = y2
  |> Kernel.-(y1)
  |> (&(&1 * &1)).()

  distance =
    x
    |> Kernel.+(y)
    |> :math.sqrt()
    |> Float.round(5)

  {neighbor, distance}
end
```

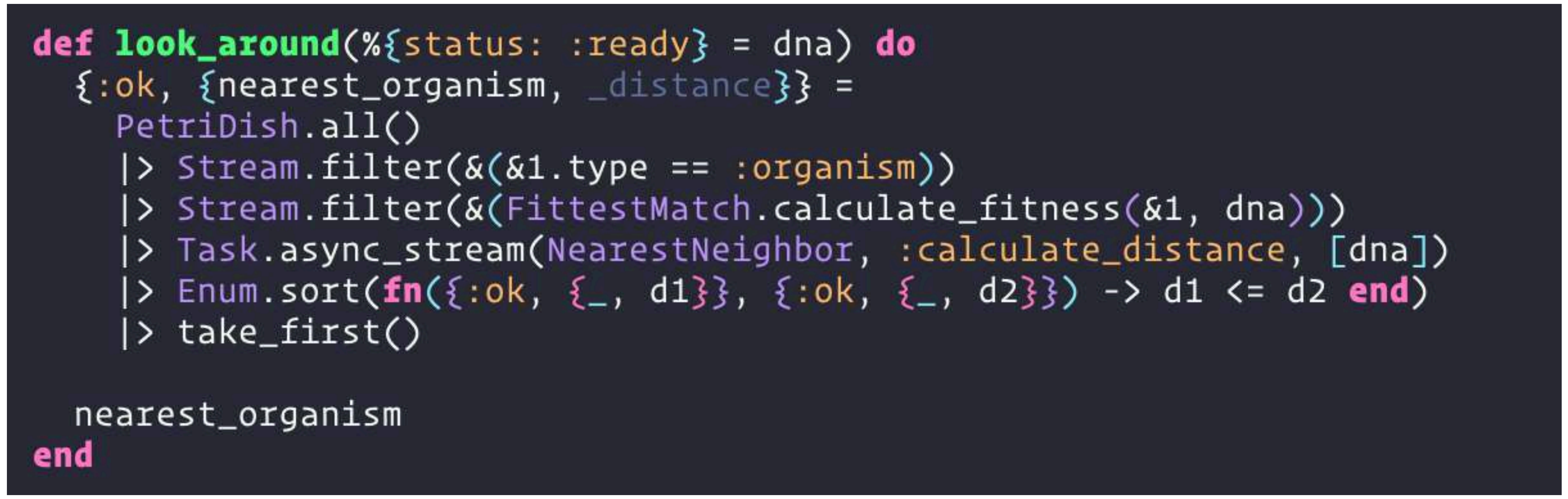
$$\sqrt{((x2 - x1)^2 + (x2 - x1)^2)}$$



lib/prototype/calculators/nearest_neighbor.ex

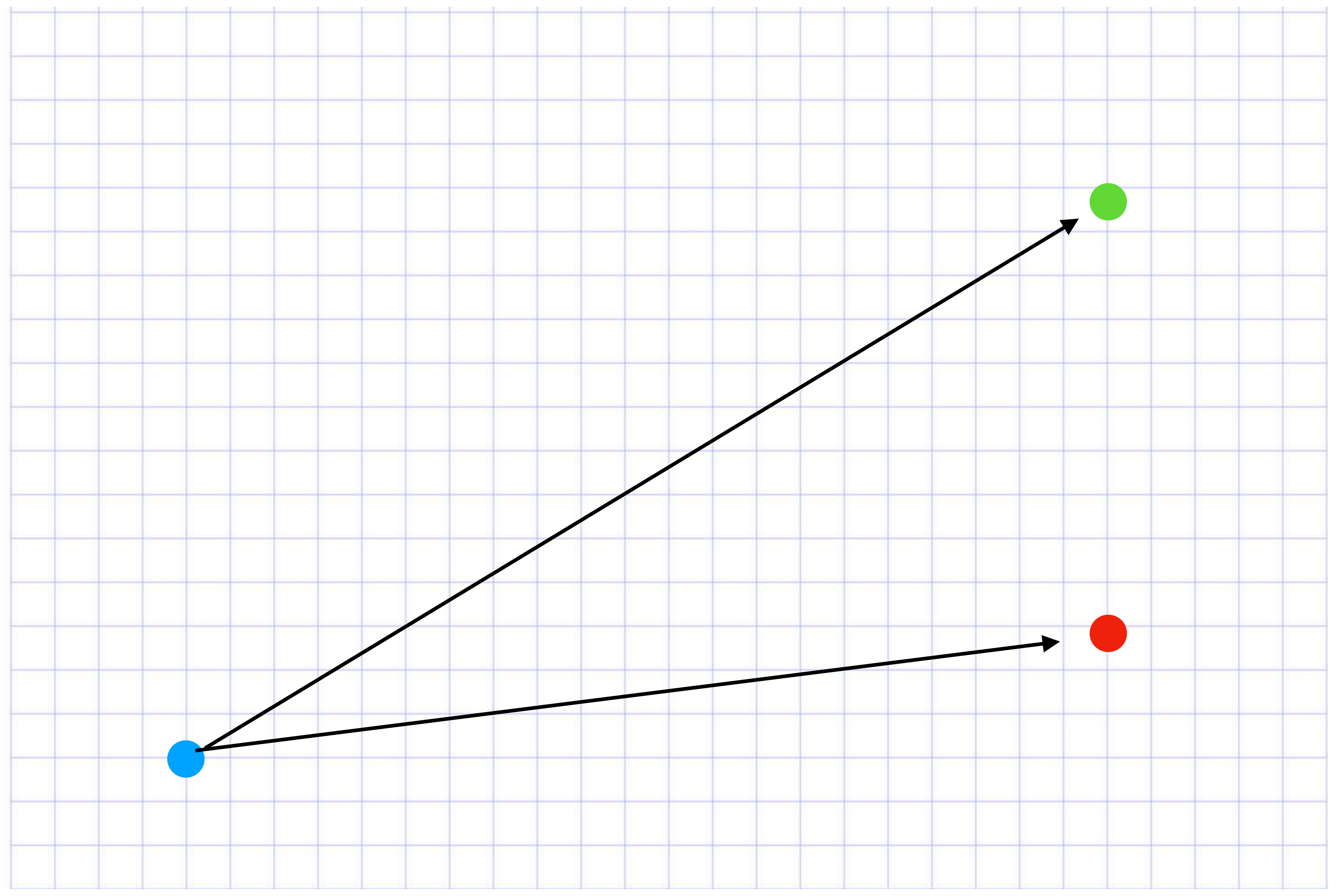
* Nearest Neighbor

Just Look Around

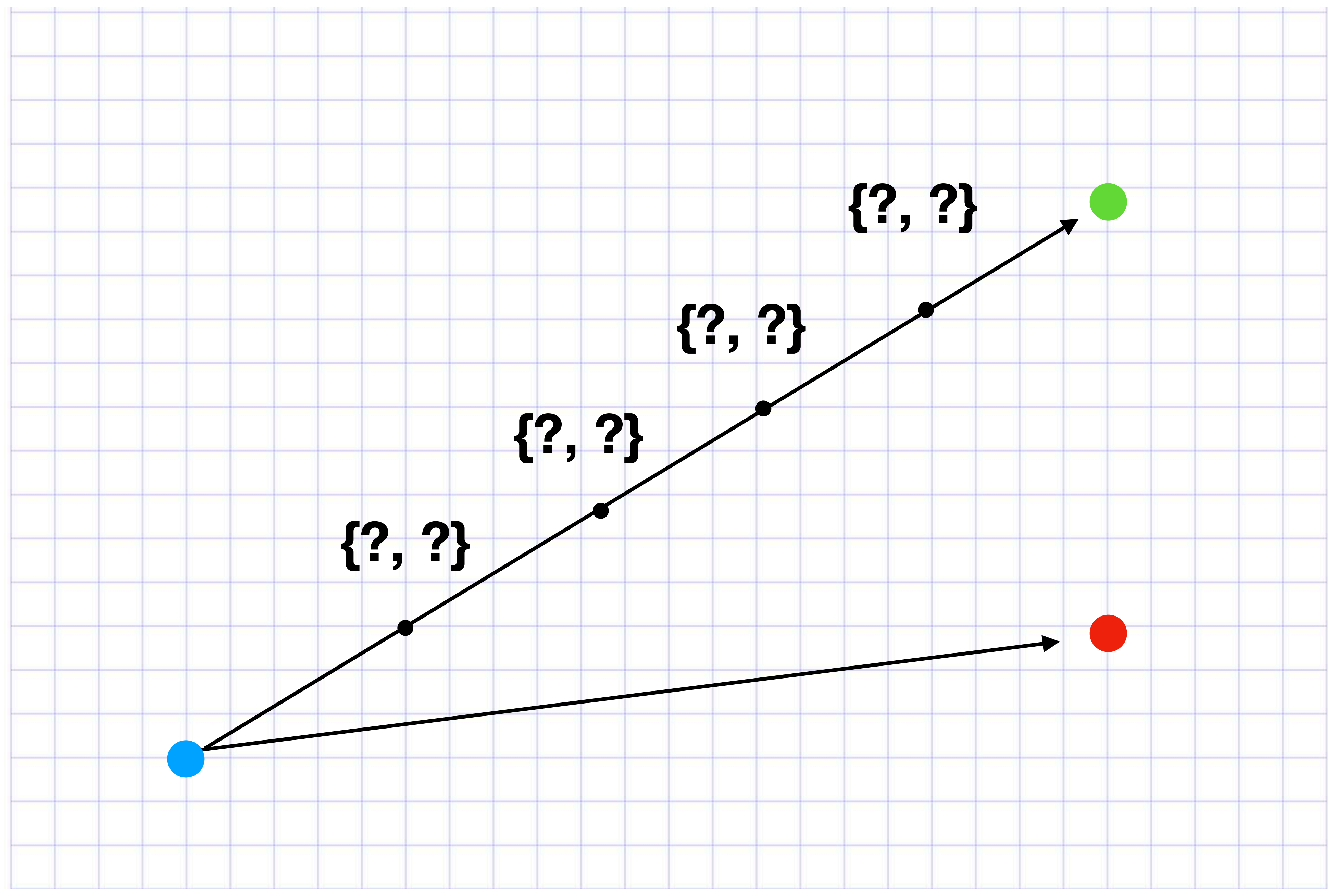


```elixir
def look_around(%{status: :ready} = dna) do
  {:ok, {nearest_organism, _distance}} =
    PetriDish.all()
    |> Stream.filter(&(&1.type == :organism))
    |> Stream.filter(&(FittestMatch.calculate_fitness(&1, dna)))
    |> Task.async_stream(NearestNeighbor, :calculate_distance, [dna])
    |> Enum.sort(fn({:ok, {_, d1}}, {:ok, {_, d2}}) -> d1 <= d2 end)
    |> take_first()

  nearest_organism
end
```

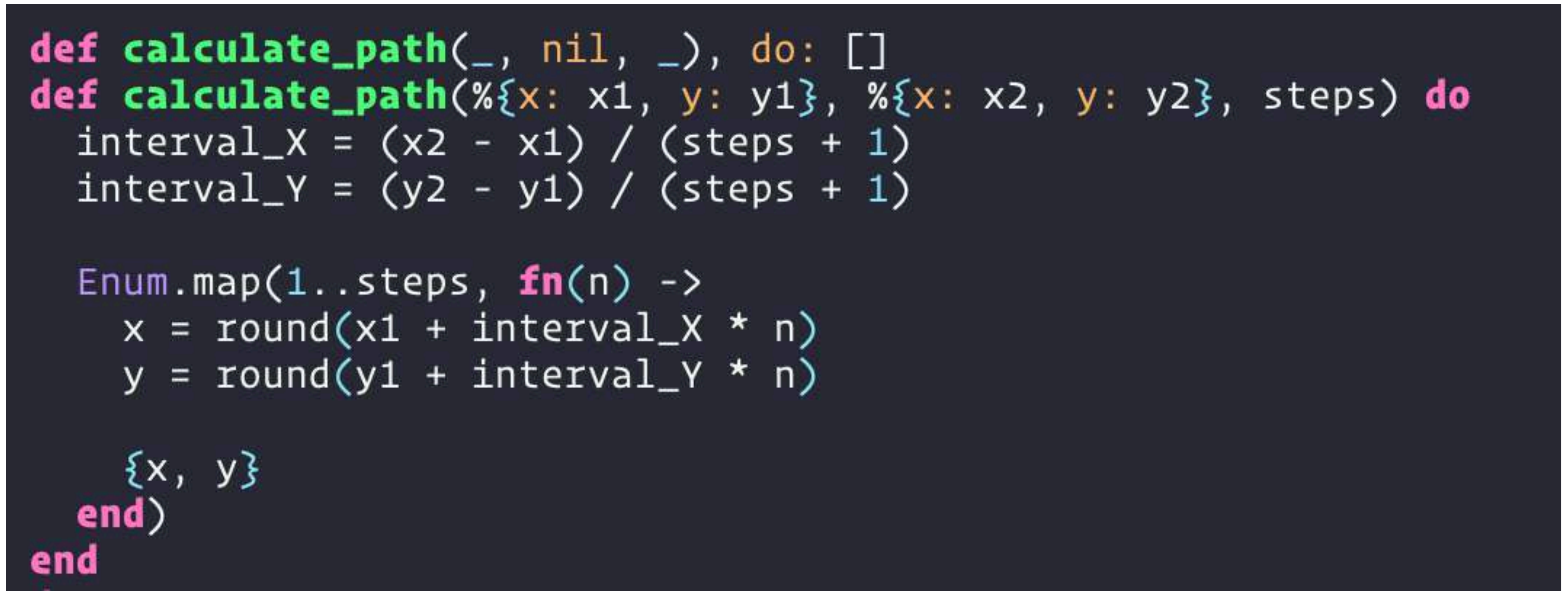
/lib/prototype/organisms/actions.ex#L109



* Trajectory

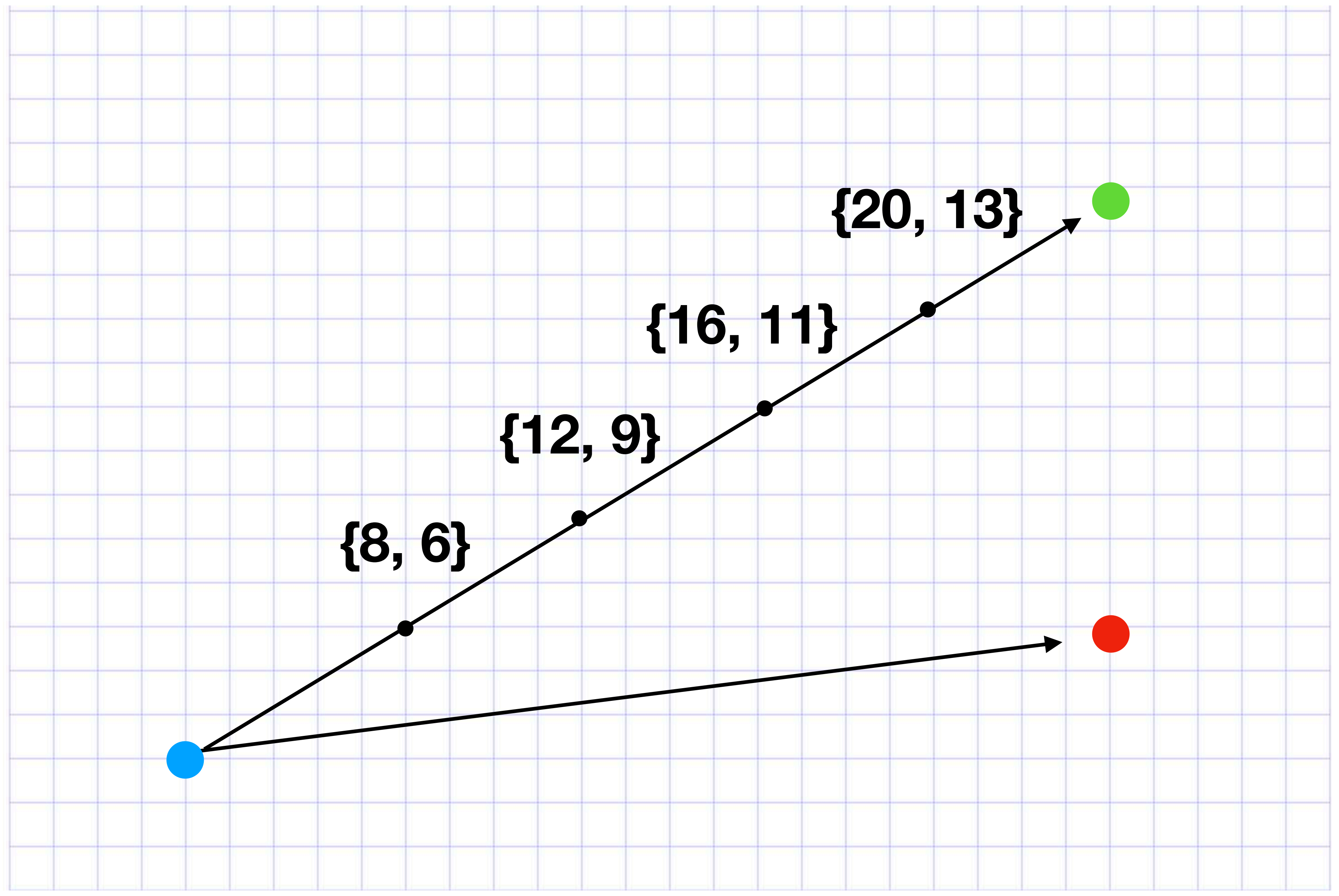


* Trajectory

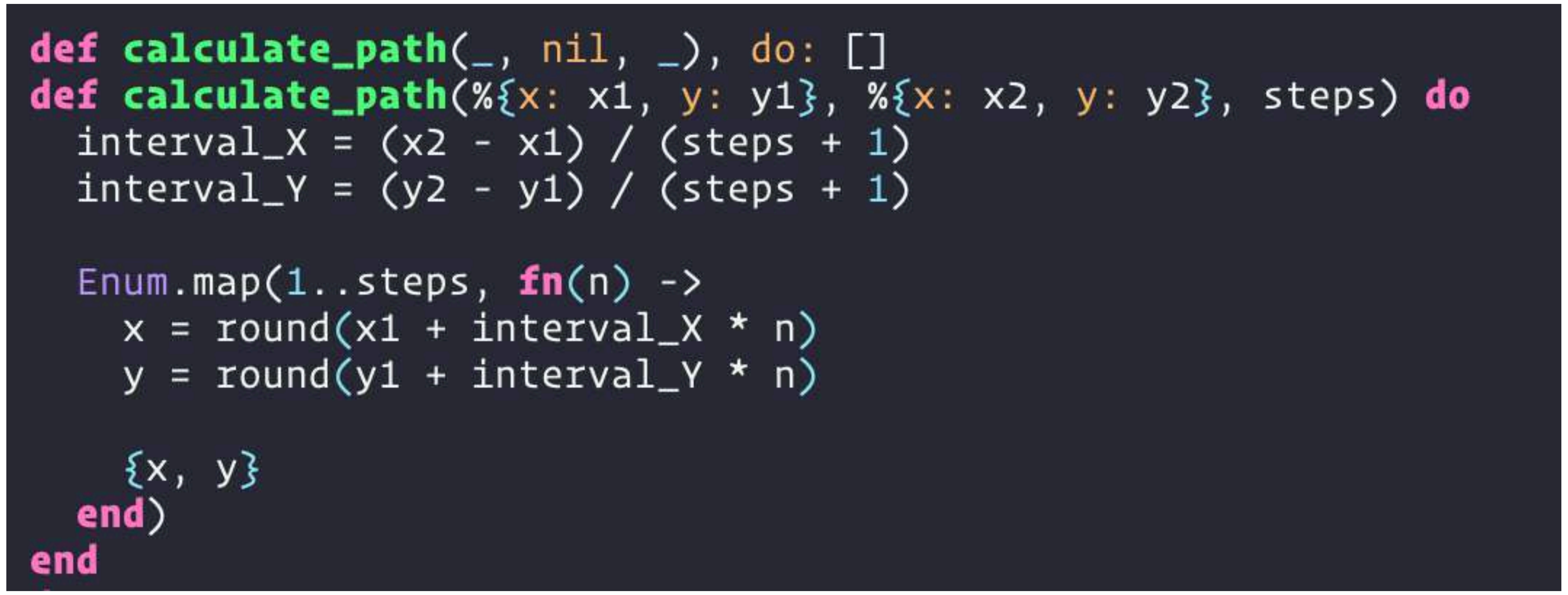


```
def calculate_path(_, nil, _), do: []
def calculate_path(%{x: x1, y: y1}, %{x: x2, y: y2}, steps) do
  interval_X = (x2 - x1) / (steps + 1)
  interval_Y = (y2 - y1) / (steps + 1)

  Enum.map(1..steps, fn(n) ->
    x = round(x1 + interval_X * n)
    y = round(y1 + interval_Y * n)

    {x, y}
  end)
end
```

lib/prototype/calculators/trajectory.ex

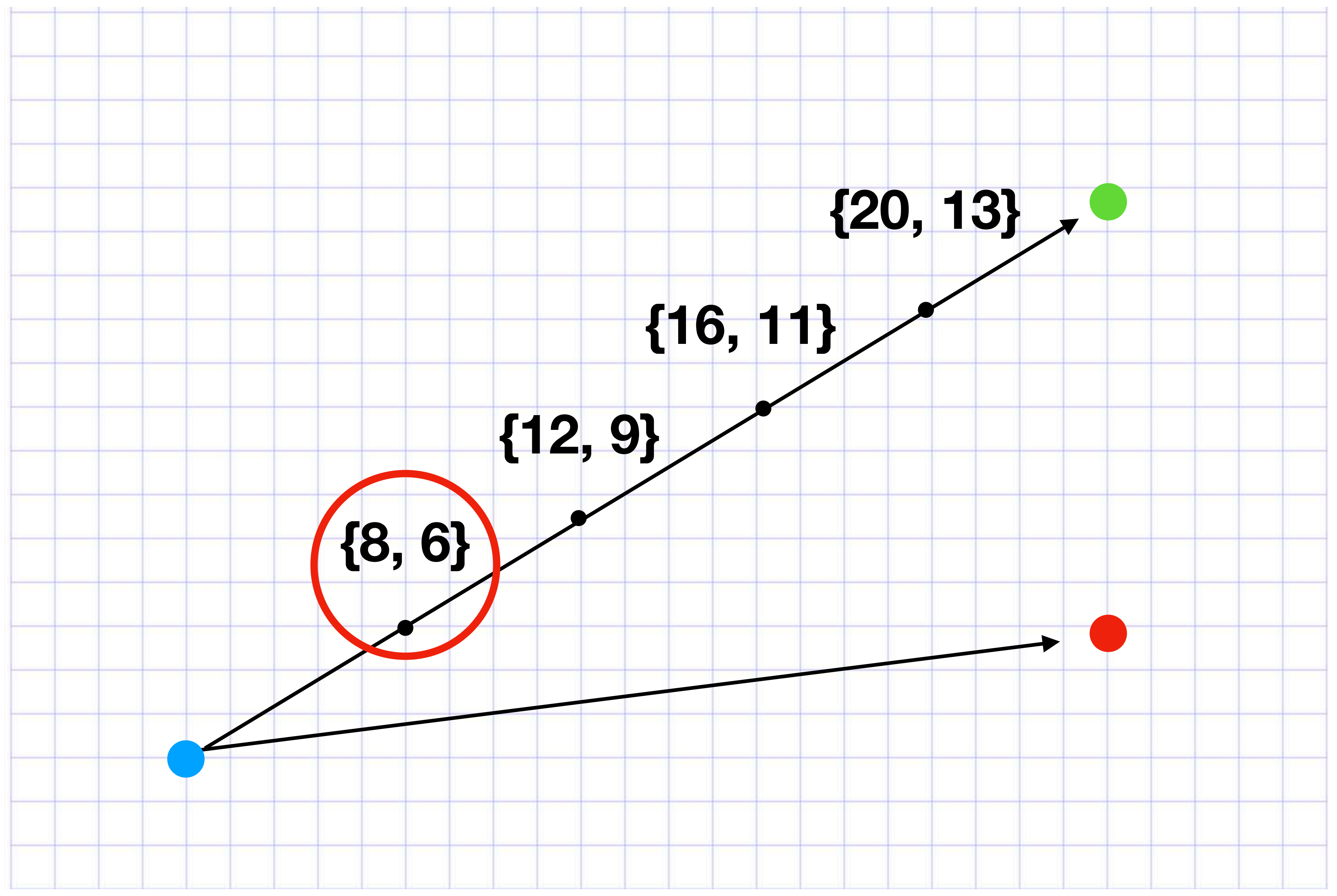


* Trajectory

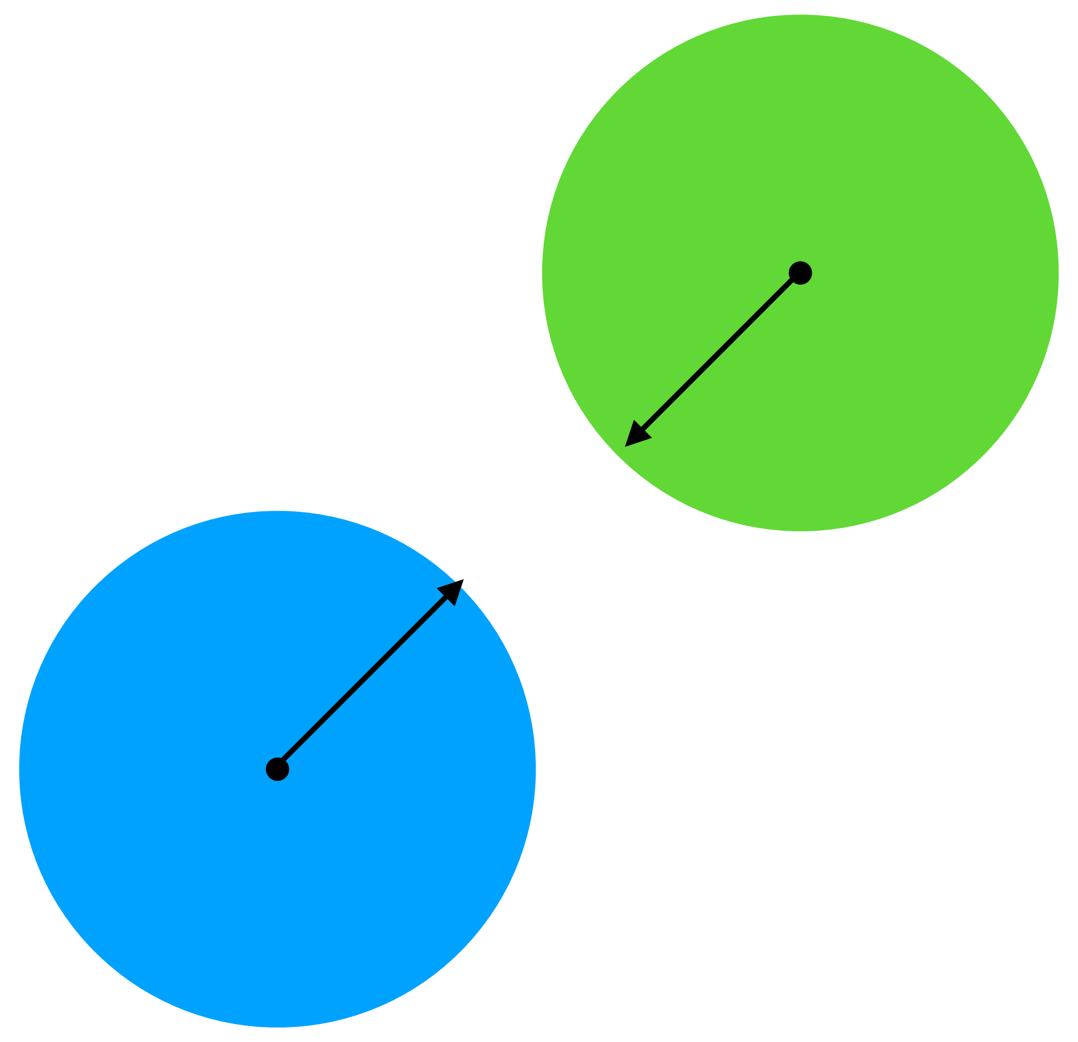


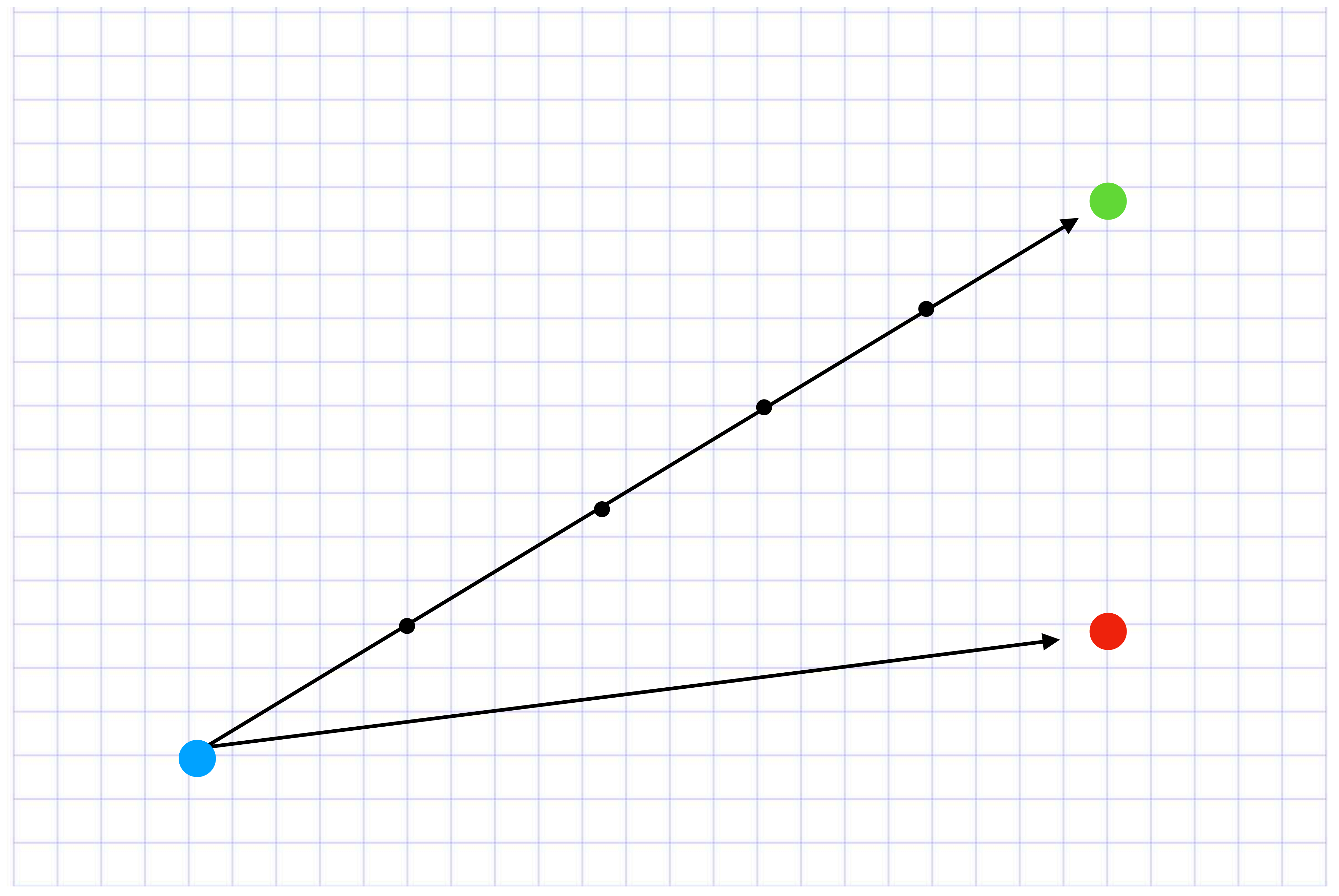
```
def calculate_path(_, nil, _), do: []
def calculate_path(%{x: x1, y: y1}, %{x: x2, y: y2}, steps) do
  interval_X = (x2 - x1) / (steps + 1)
  interval_Y = (y2 - y1) / (steps + 1)

  Enum.map(1..steps, fn(n) ->
    x = round(x1 + interval_X * n)
    y = round(y1 + interval_Y * n)

    {x, y}
  end)
end
```

lib/prototype/calculators/trajectory.ex

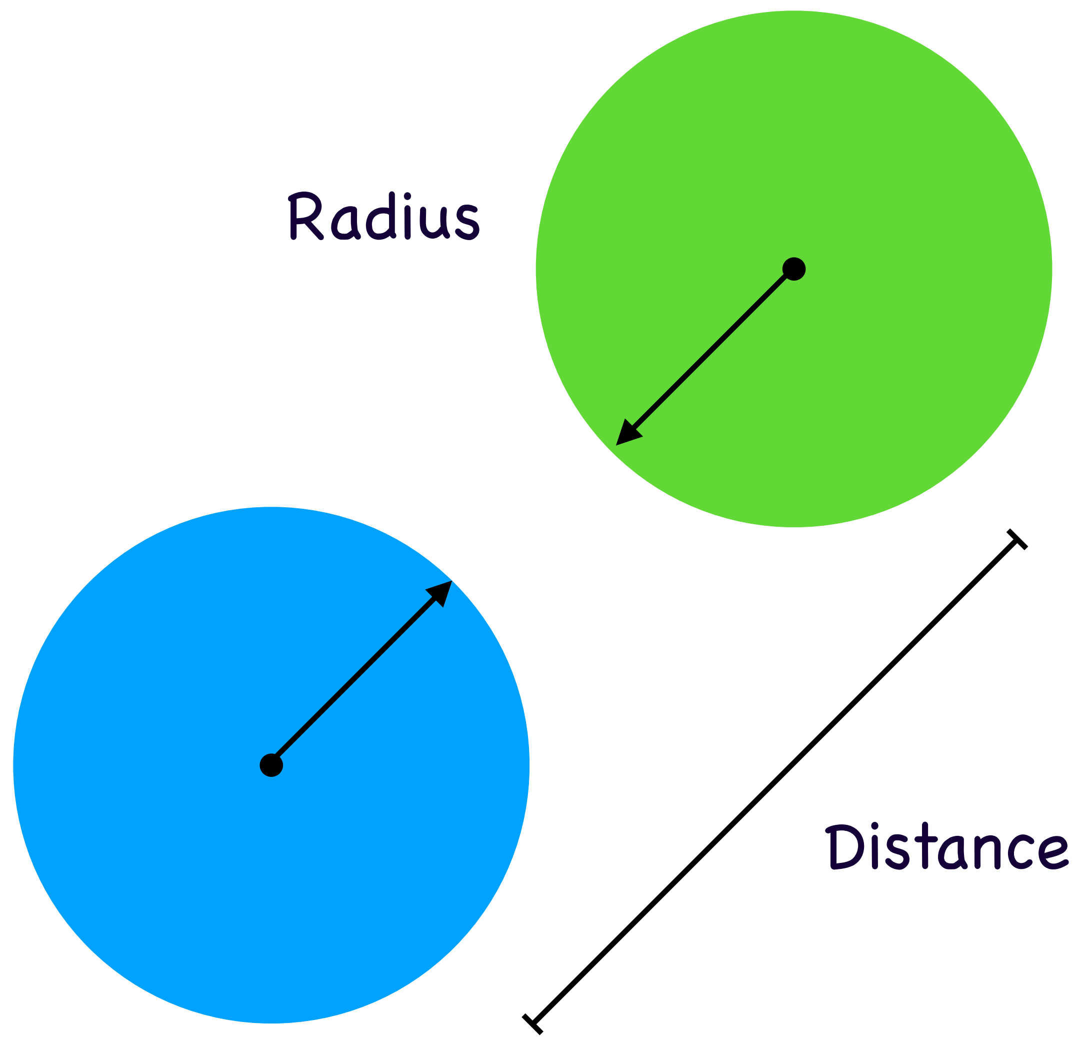


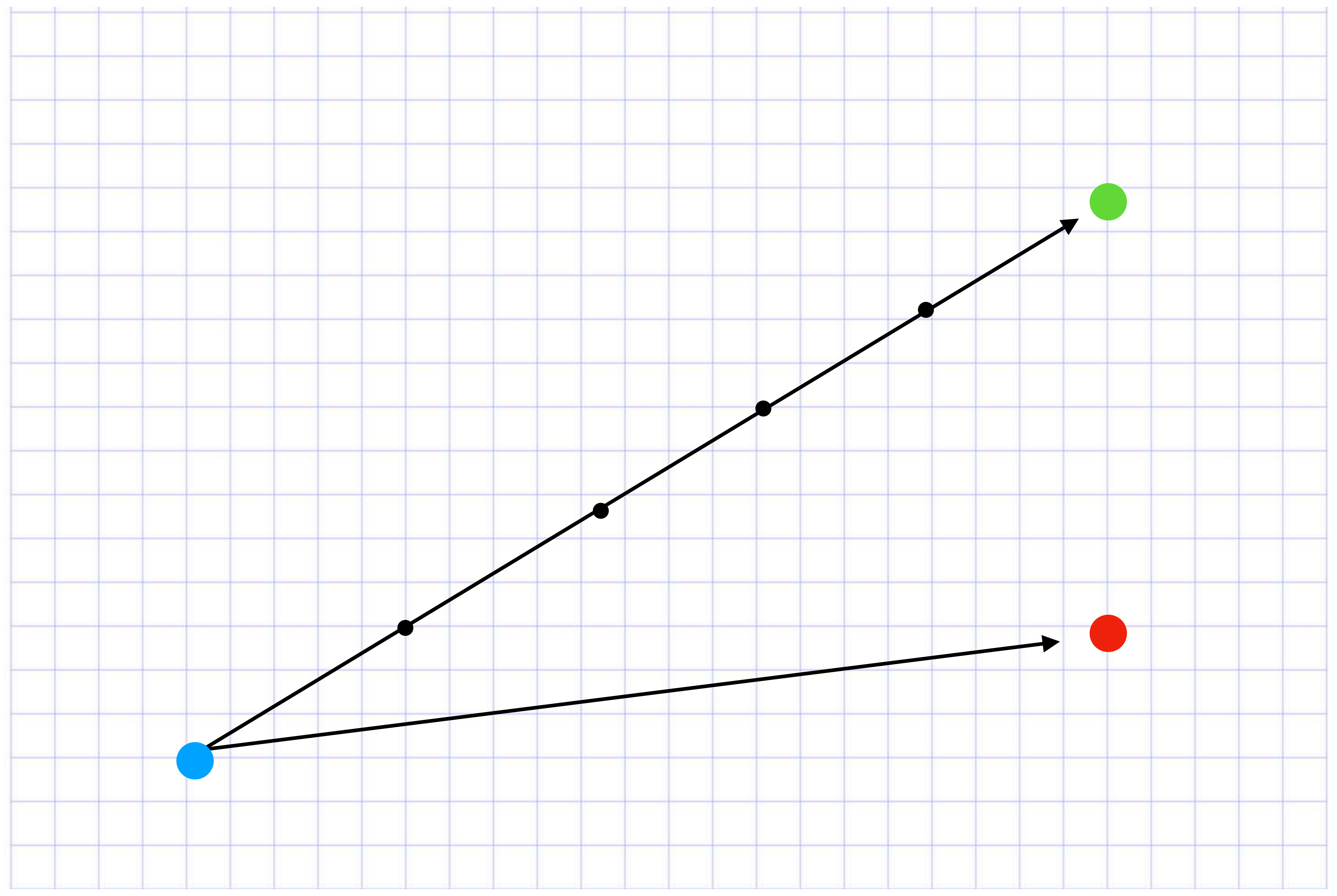
* Collision Detection



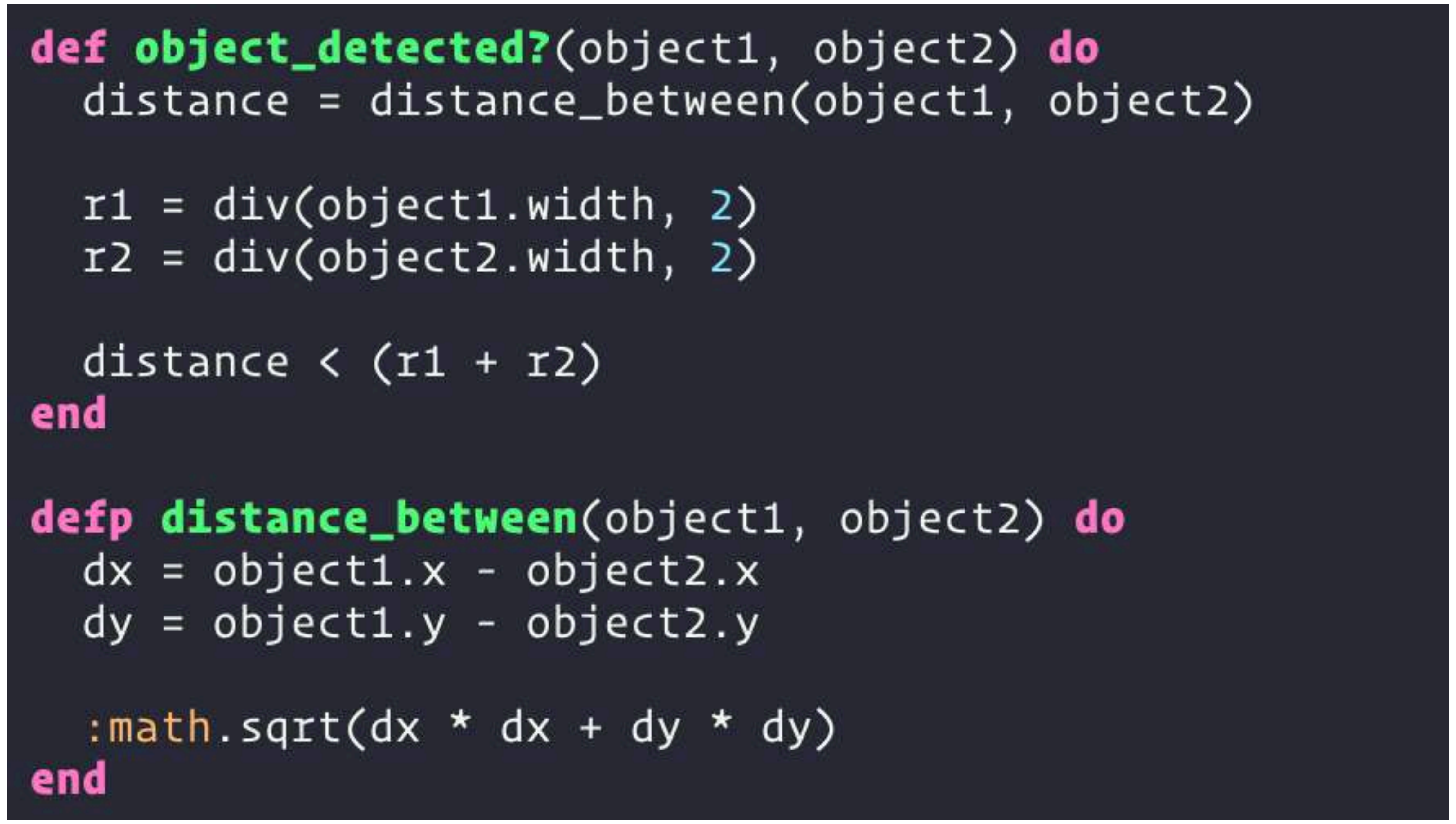


* Collision Detection

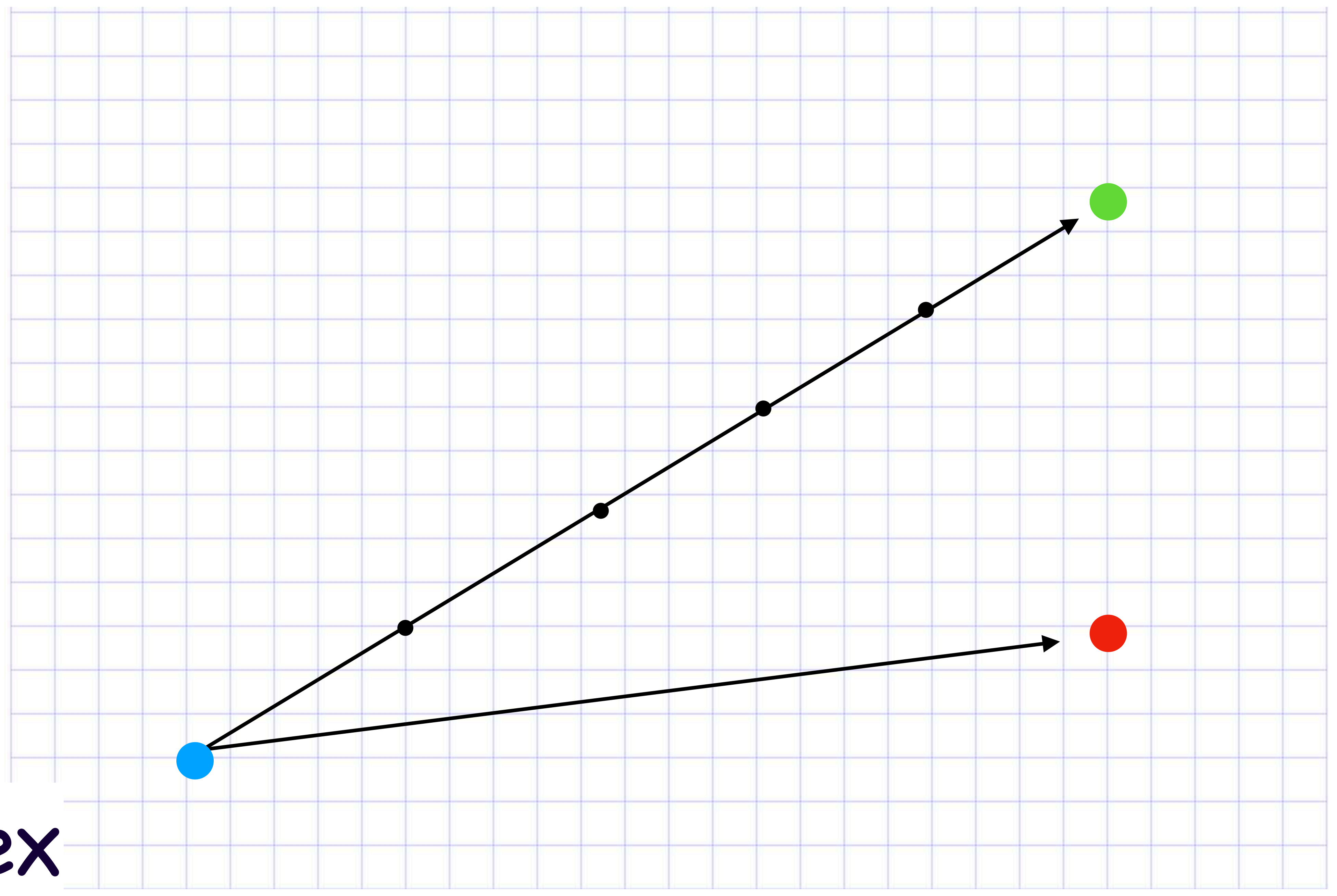




* Collision Detection

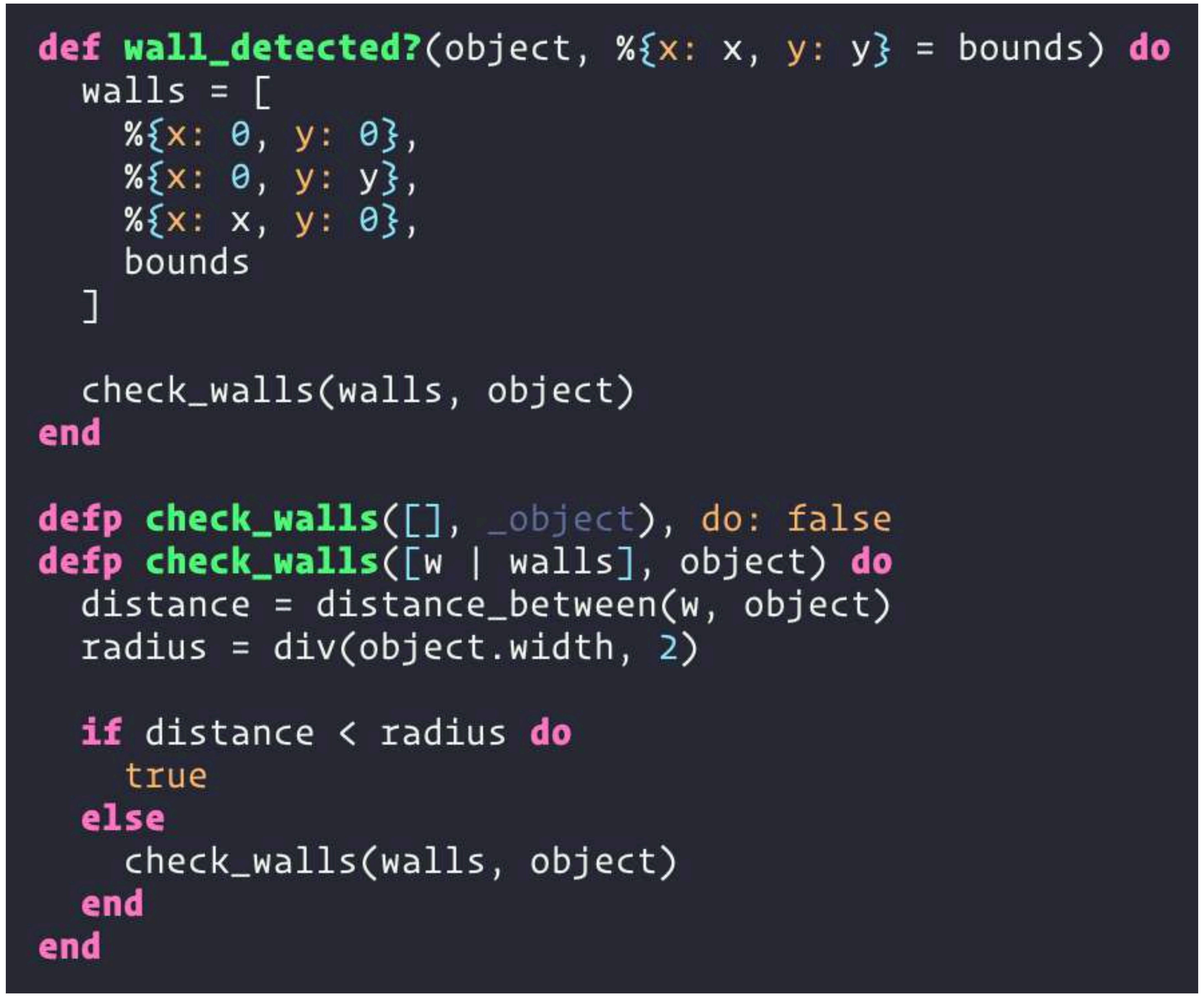


```elixir
def object_detected?(object1, object2) do
  distance = distance_between(object1, object2)

  r1 = div(object1.width, 2)
  r2 = div(object2.width, 2)

  distance < (r1 + r2)
end

defp distance_between(object1, object2) do
  dx = object1.x - object2.x
  dy = object1.y - object2.y

  :math.sqrt(dx * dx + dy * dy)
end
```



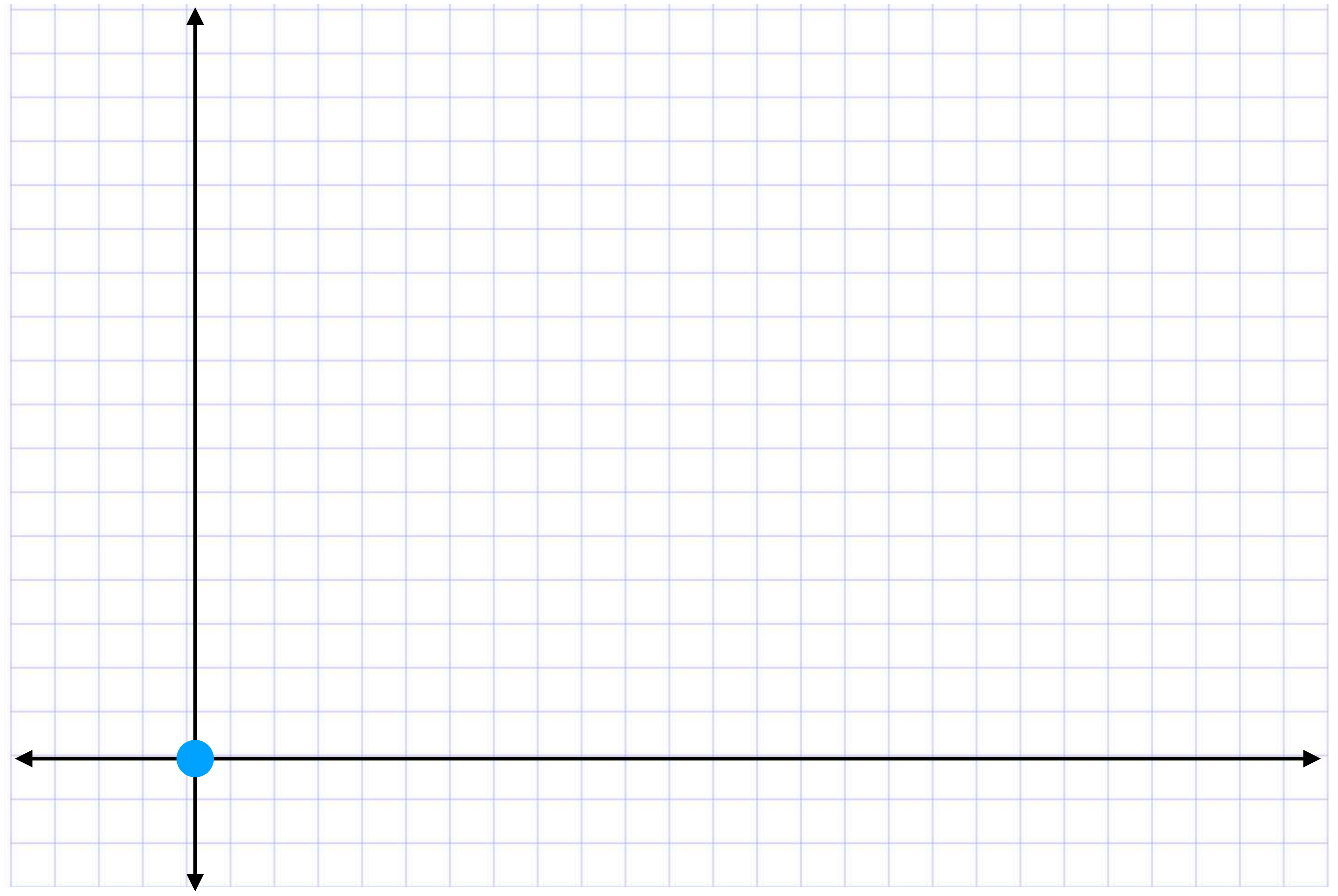
lib/prototype/calculators/collision_detection.ex

* Collision Detection

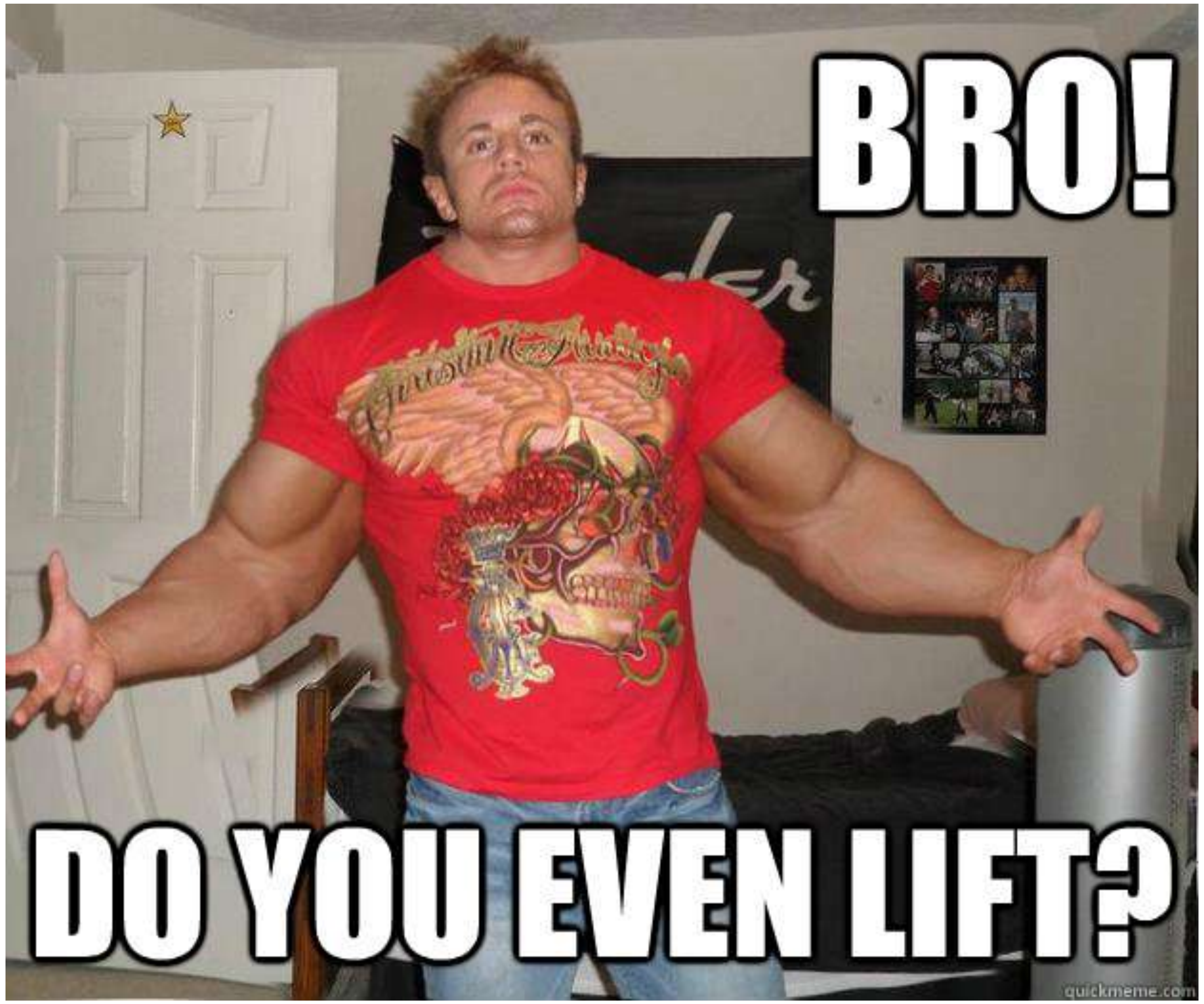


```elixir
def wall_detected?(object, %{x: x, y: y} = bounds) do
  walls = [
    %{x: 0, y: 0},
    %{x: 0, y: y},
    %{x: x, y: 0},
    bounds
  ]

  check_walls(walls, object)
end

defp check_walls([], _object), do: false
defp check_walls([w | walls], object) do
  distance = distance_between(w, object)
  radius = div(object.width, 2)

  if distance < radius do
    true
  else
    check_walls(walls, object)
  end
end
```

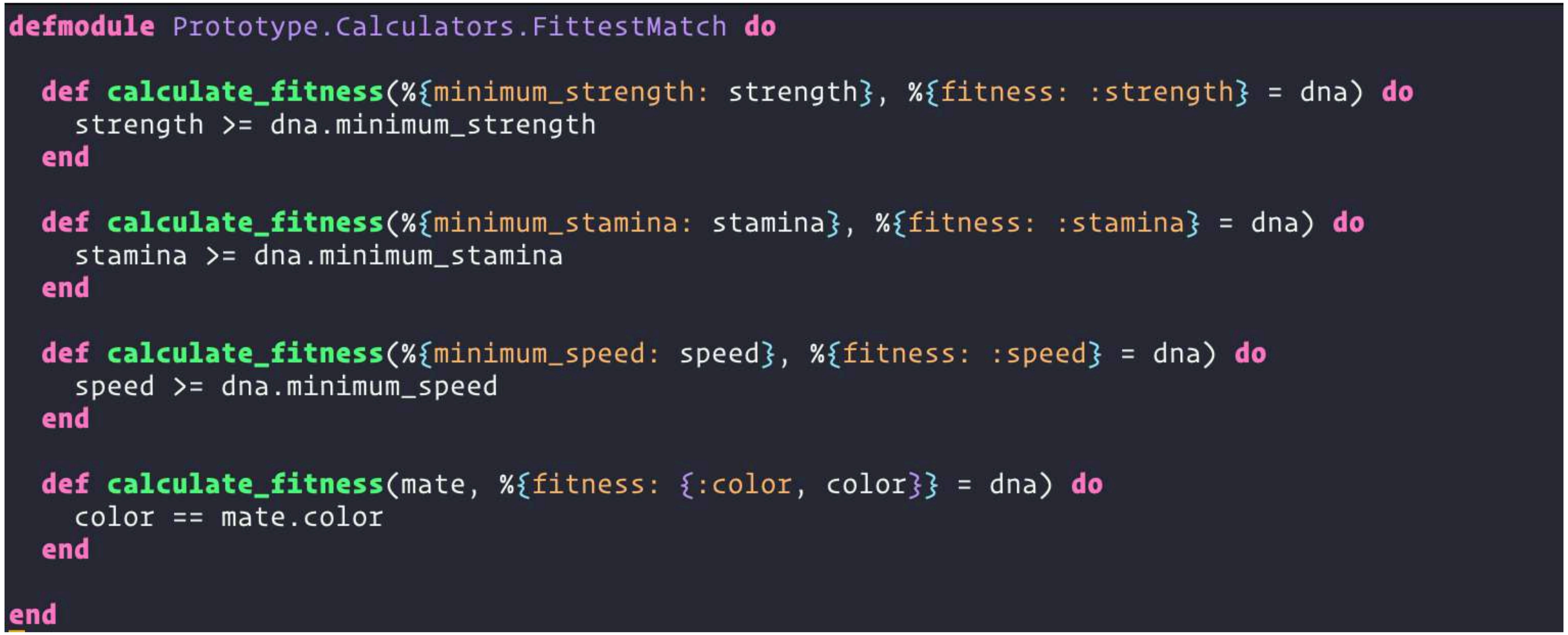
What About the Walls?



* Fitness?



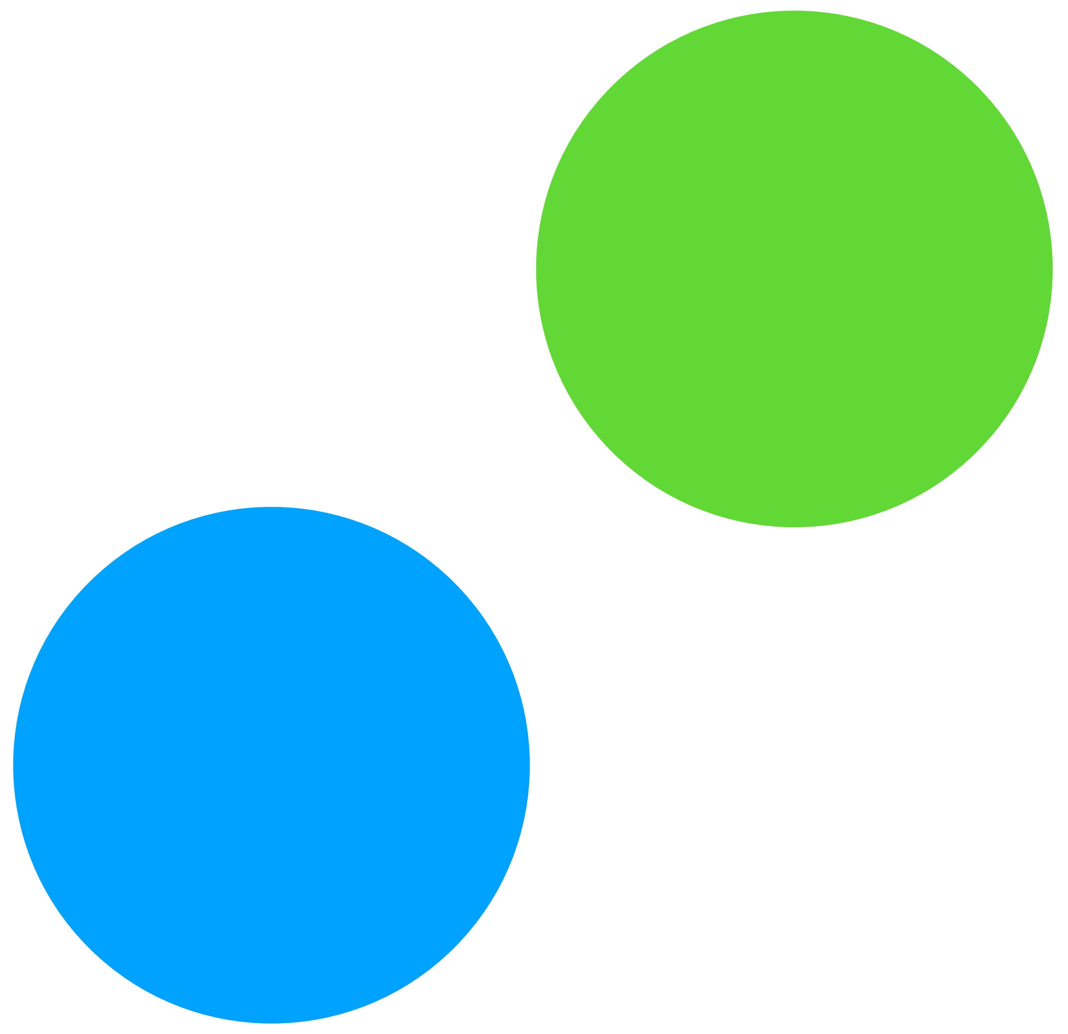
* Fitness



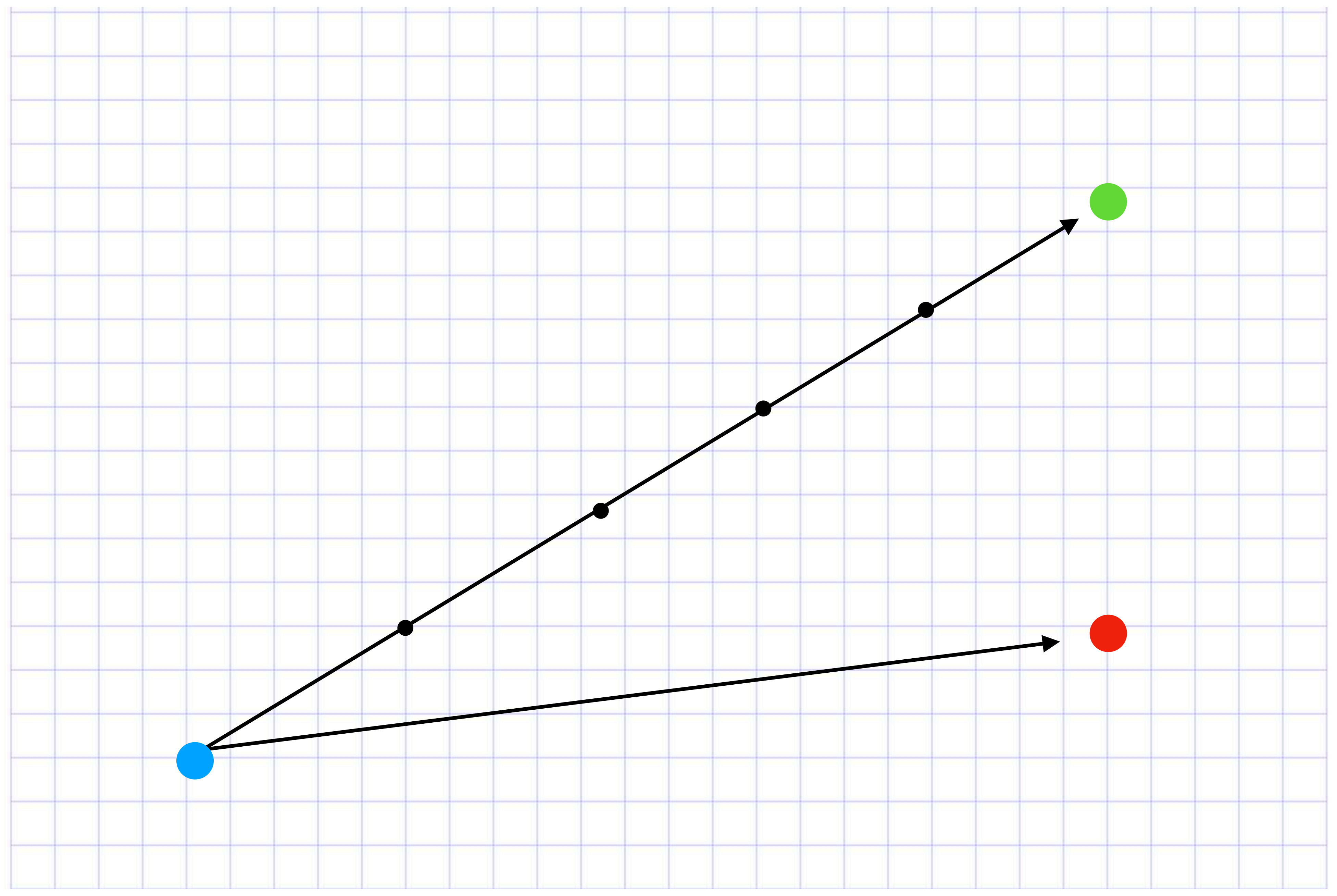
```elixir
defmodule Prototype.Calculators.FittestMatch do

  def calculate_fitness(%{minimum_strength: strength}, %{fitness: :strength} = dna) do
    strength >= dna.minimum_strength
  end

  def calculate_fitness(%{minimum_stamina: stamina}, %{fitness: :stamina} = dna) do
    stamina >= dna.minimum_stamina
  end

  def calculate_fitness(%{minimum_speed: speed}, %{fitness: :speed} = dna) do
    speed >= dna.minimum_speed
  end

  def calculate_fitness(mate, %{fitness: {:color, color}} = dna) do
    color == mate.color
  end

end
```

lib/prototype/calculators/fittest_match.ex

* Assigning Traits



Green or Blue?



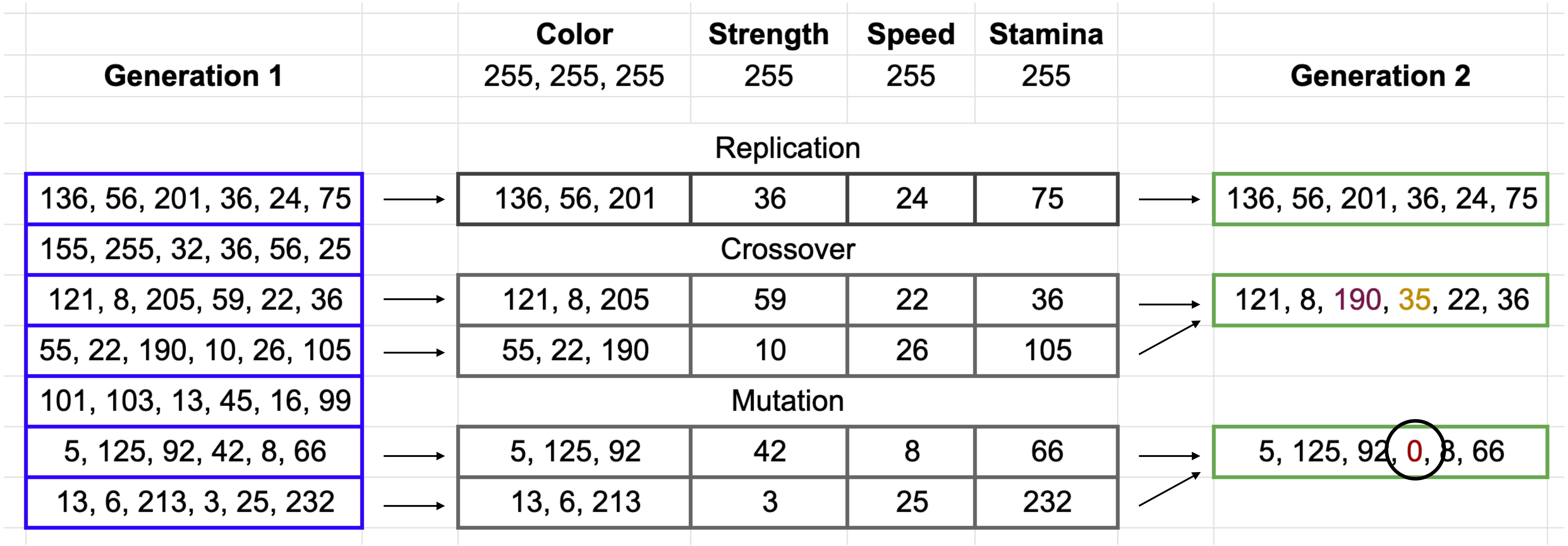
* Assigning Traits

		Color	**Strength**	**Speed**	**Stamina**		
Generation 1		255, 255, 255	255	255	255		**Generation 2**
		Replication					
136, 56, 201, 36, 24, 75	→	136, 56, 201	36	24	75	→	136, 56, 201, 36, 24, 75
155, 255, 32, 36, 56, 25		Crossover					
121, 8, 205, 59, 22, 36		121, 8, 205	59	22	36		121, 8, 190, 35, 22, 36
55, 22, 190, 10, 26, 105		55, 22, 190	10	26	105		
101, 103, 13, 45, 16, 99		Mutation					
5, 125, 92, 42, 8, 66		5, 125, 92	42	8	66		5, 125, 92, 0, 8, 66
13, 6, 213, 3, 25, 232		13, 6, 213	3	25	232		

* Assigning Traits

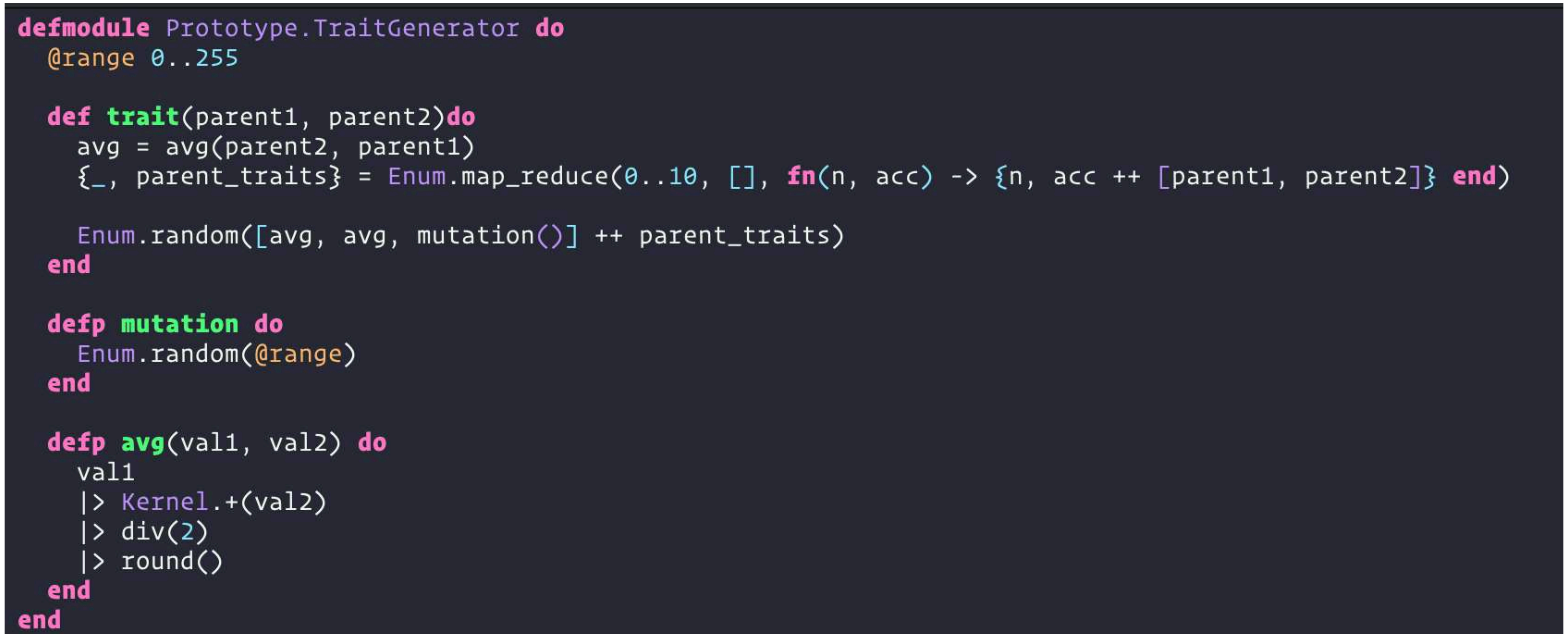
		Color	Strength	Speed	Stamina		
Generation 1		255, 255, 255	255	255	255		**Generation 2**
		Replication					
136, 56, 201, 36, 24, 75	→	136, 56, 201	36	24	75	→	136, 56, 201, 36, 24, 75
155, 255, 32, 36, 56, 25		Crossover					
121, 8, 205, 59, 22, 36	→	121, 8, 205	59	22	36	→	121, 8, 190, 35, 22, 36
55, 22, 190, 10, 26, 105	→	55, 22, 190	10	26	105	↗	
101, 103, 13, 45, 16, 99		Mutation					
5, 125, 92, 42, 8, 66		5, 125, 92	42	8	66		5, 125, 92, 0, 8, 66
13, 6, 213, 3, 25, 232		13, 6, 213	3	25	232		

* Assigning Traits



Generation 1		Color	Strength	Speed	Stamina		Generation 2
		255, 255, 255	255	255	255		
		Replication					
136, 56, 201, 36, 24, 75	→	136, 56, 201	36	24	75	→	136, 56, 201, 36, 24, 75
155, 255, 32, 36, 56, 25		Crossover					
121, 8, 205, 59, 22, 36	→	121, 8, 205	59	22	36	→	121, 8, 190, 35, 22, 36
55, 22, 190, 10, 26, 105	→	55, 22, 190	10	26	105	↗	
101, 103, 13, 45, 16, 99		Mutation					
5, 125, 92, 42, 8, 66	→	5, 125, 92	42	8	66	→	5, 125, 92, 0, 8, 66
13, 6, 213, 3, 25, 232	→	13, 6, 213	3	25	232	↗	

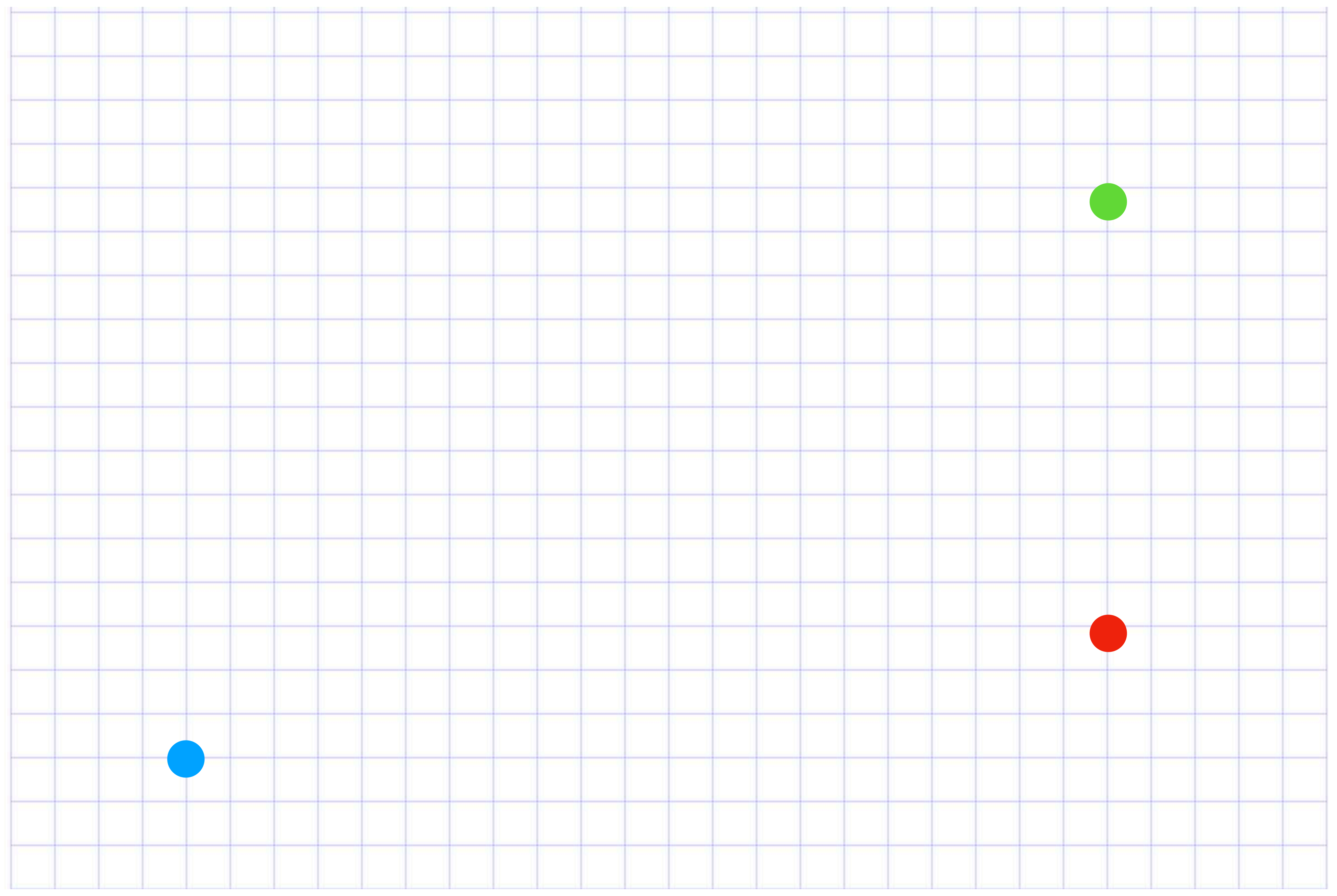
* Assigning Traits



```
defmodule Prototype.TraitGenerator do
  @range 0..255

  def trait(parent1, parent2)do
    avg = avg(parent2, parent1)
    {_, parent_traits} = Enum.map_reduce(0..10, [], fn(n, acc) -> {n, acc ++ [parent1, parent2]} end)

    Enum.random([avg, avg, mutation()] ++ parent_traits)
  end

  defp mutation do
    Enum.random(@range)
  end

  defp avg(val1, val2) do
    val1
    |> Kernel.+(val2)
    |> div(2)
    |> round()
  end
end
```

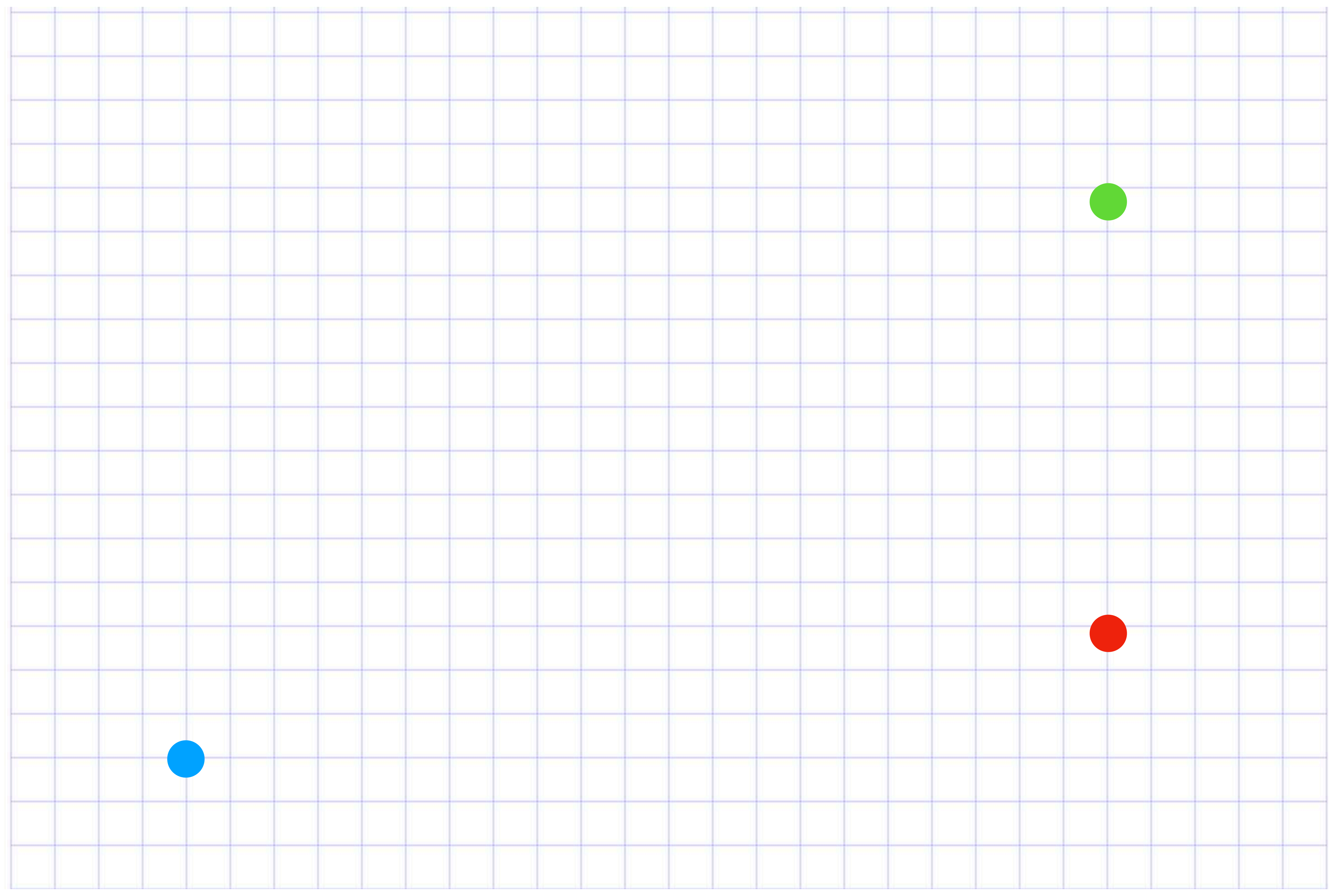
lib/prototype/trait_generator.ex

Hello, Little World! (again)

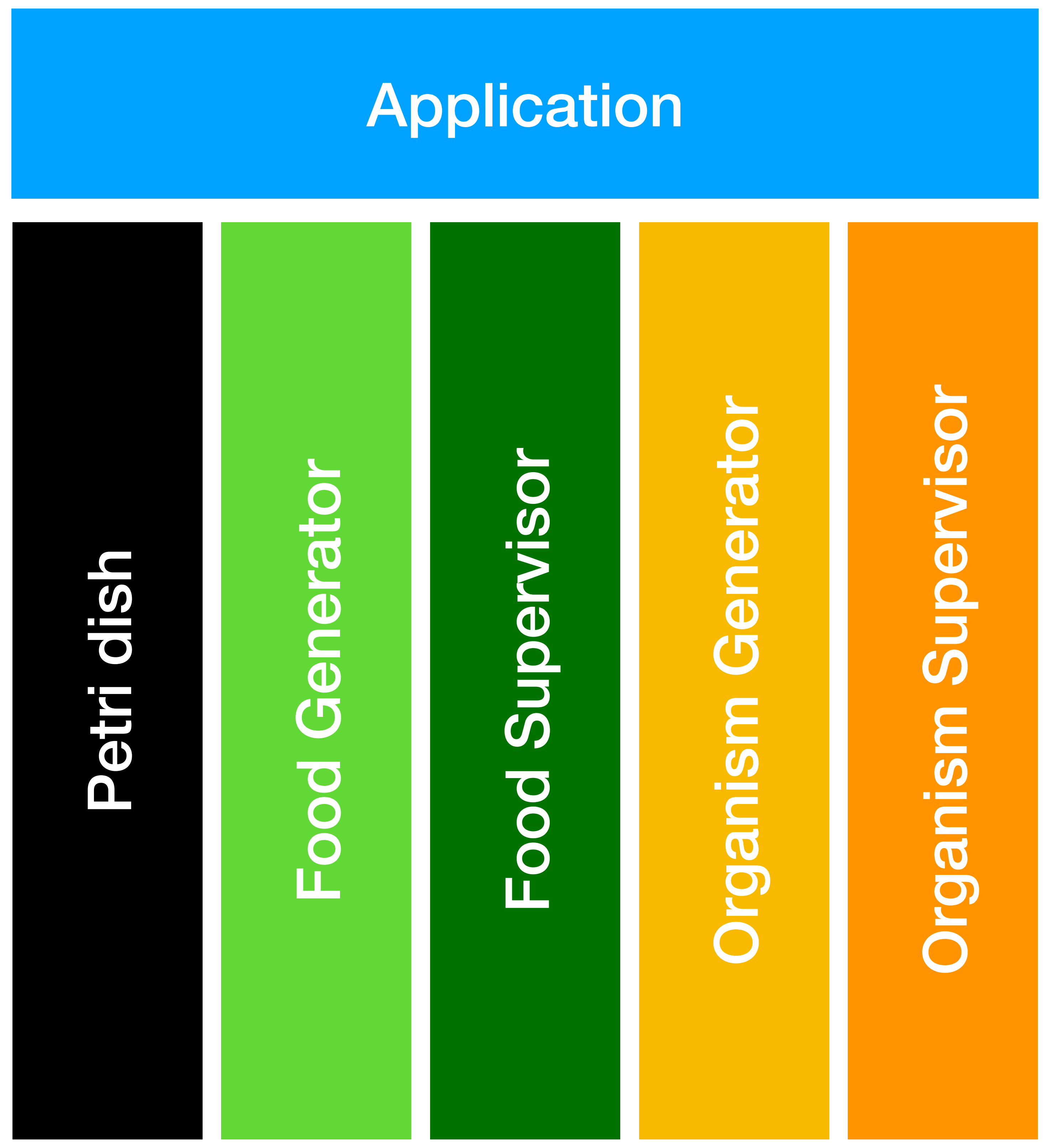


Hello, Little World! (again)

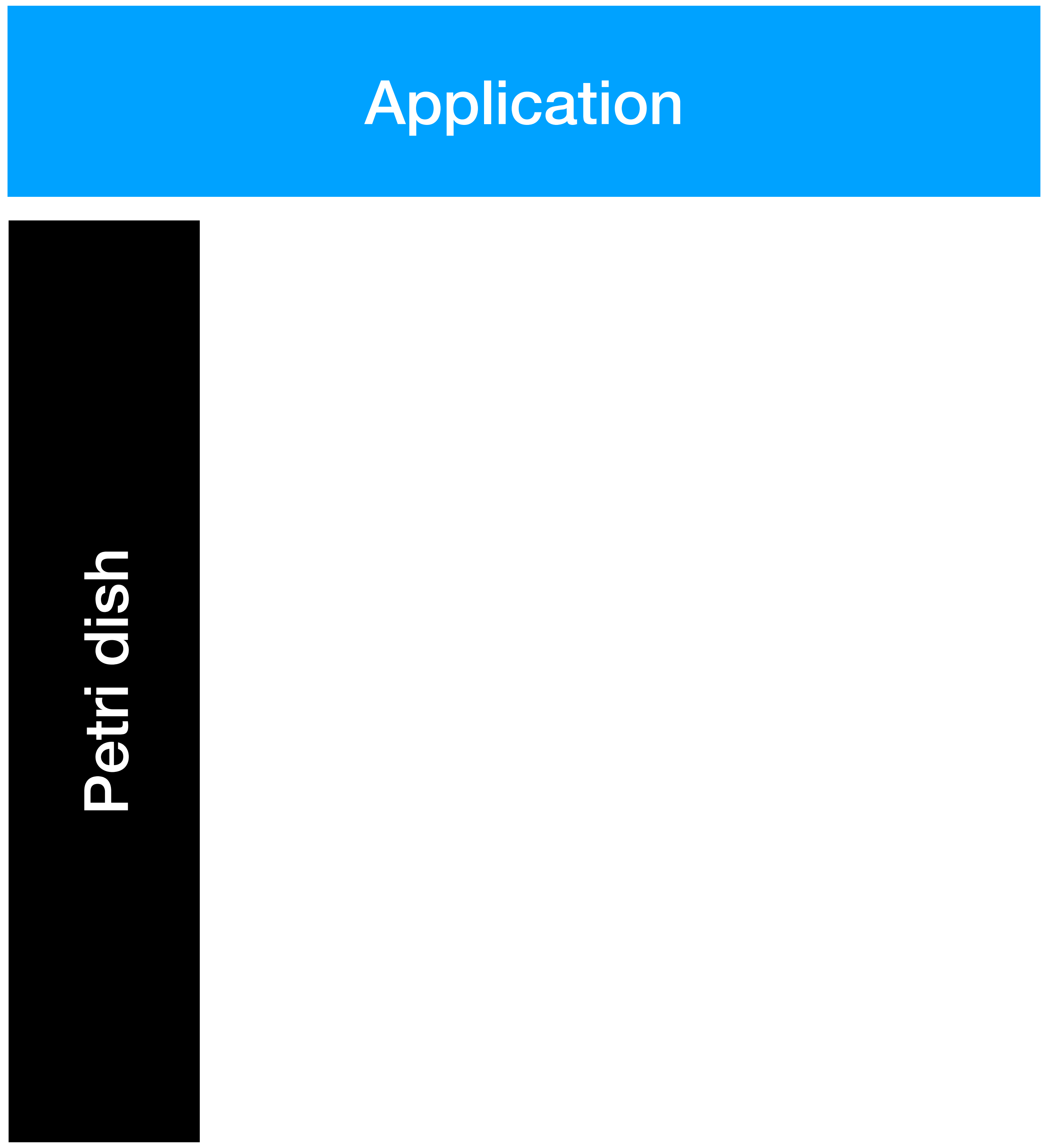
No OTP, yet?



*Building the Playground



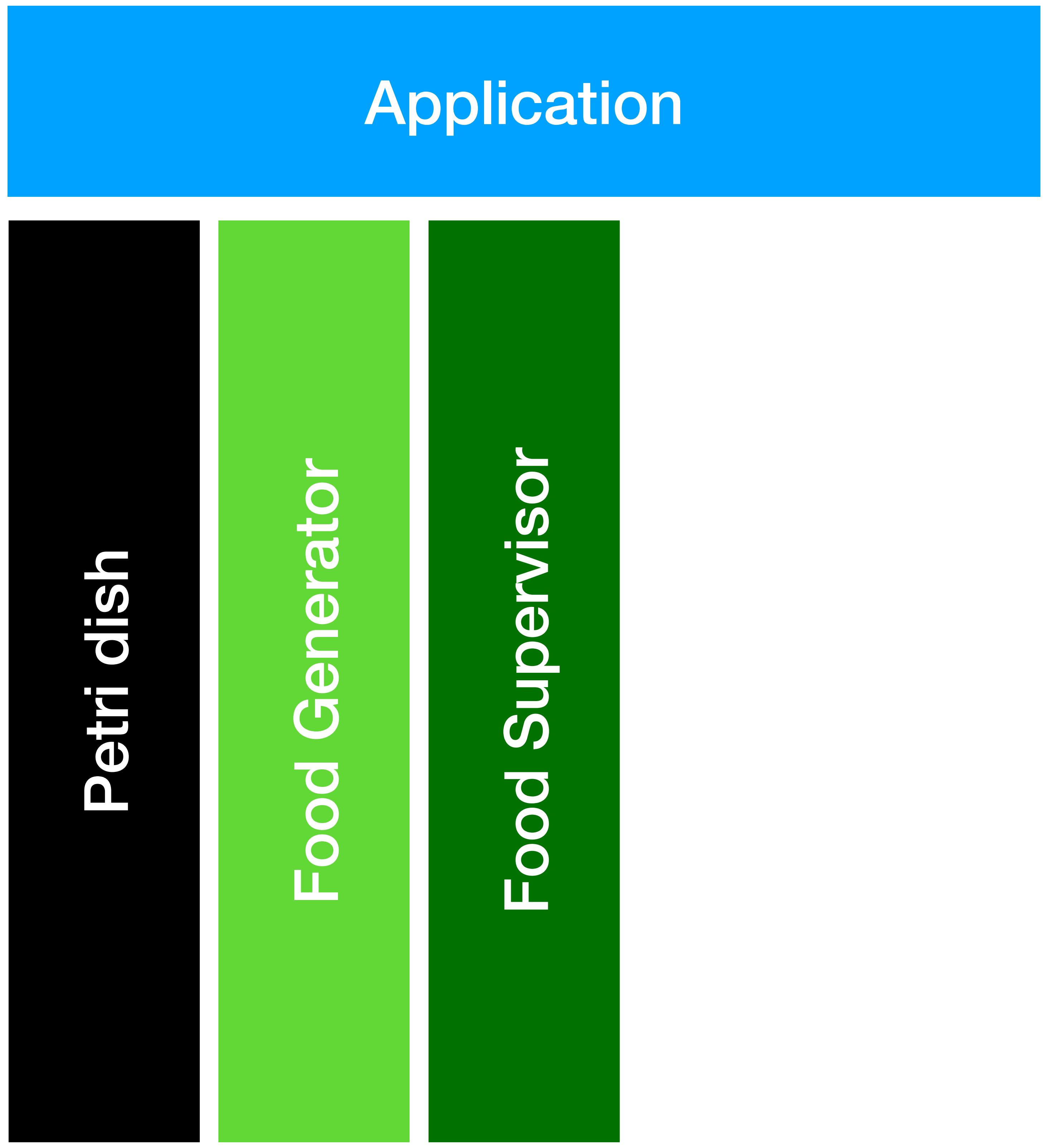
*Building the Playground



all/0	Gets all the objects in state
draw/1	Adds object to state
remove/1	Removes an object from state

lib/prototype/petri_dish.ex

*Building the Playground



begin_generation/1

change_frequency/1

change_bounds/1

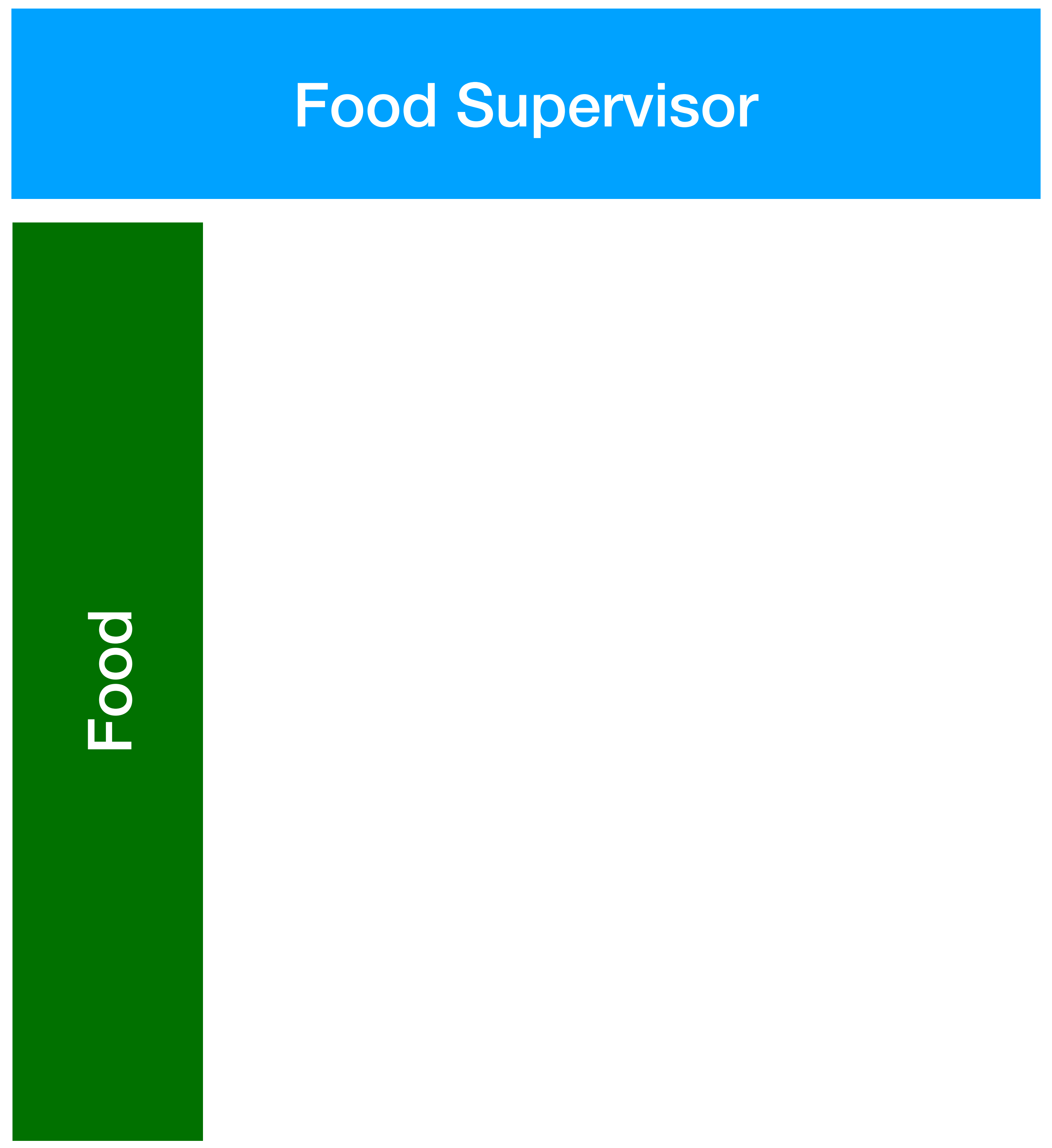
change_delay/1

change_max_count/1

*generate_food/1

lib/prototype/food_generator.ex

*Building the Playground

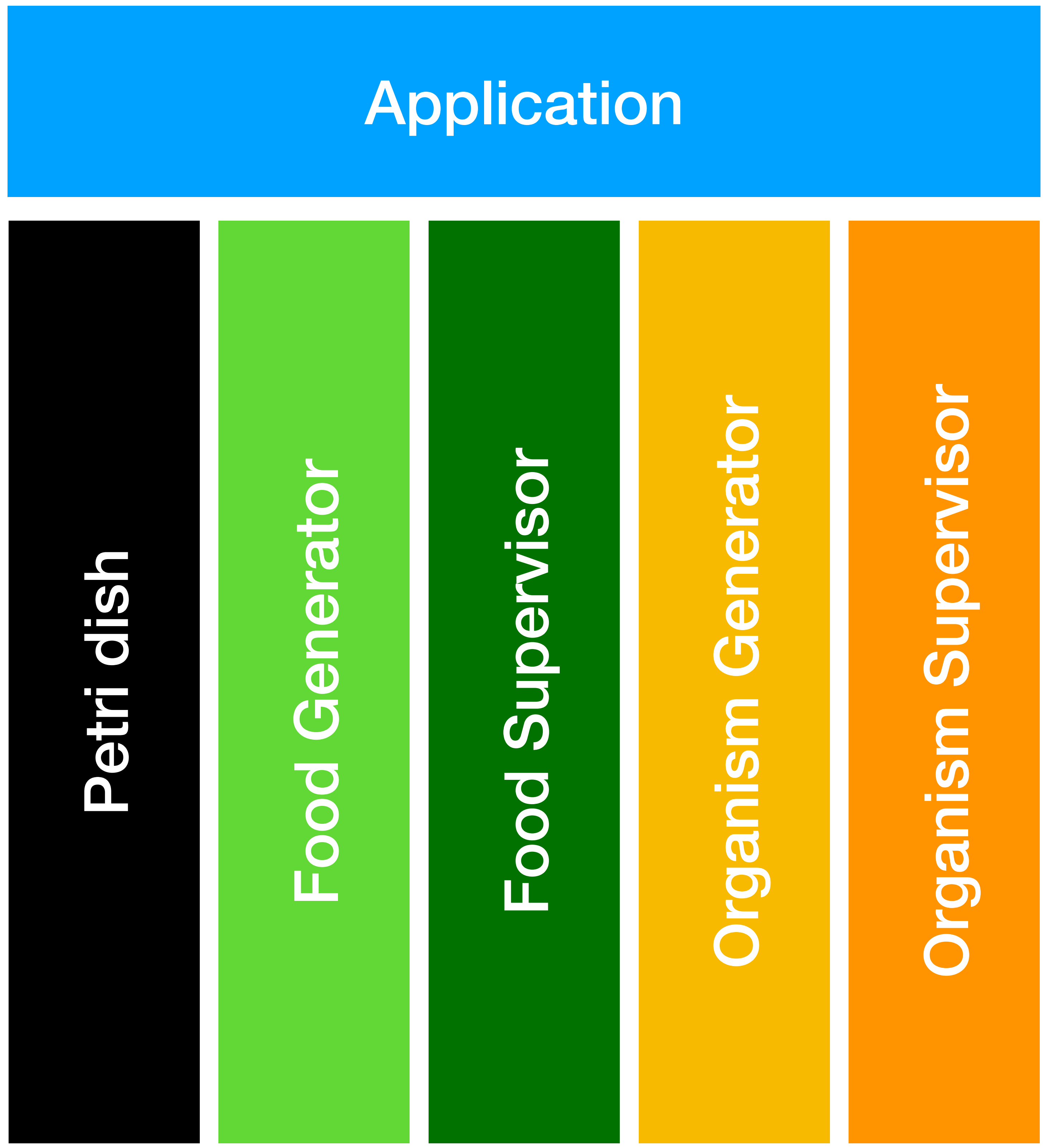


lib/prototype/food.ex

consumed/1

*set_timer/1

*Building the Playground



begin_generation/1

change_fitness/1

change_bounds/1

change_delay/1

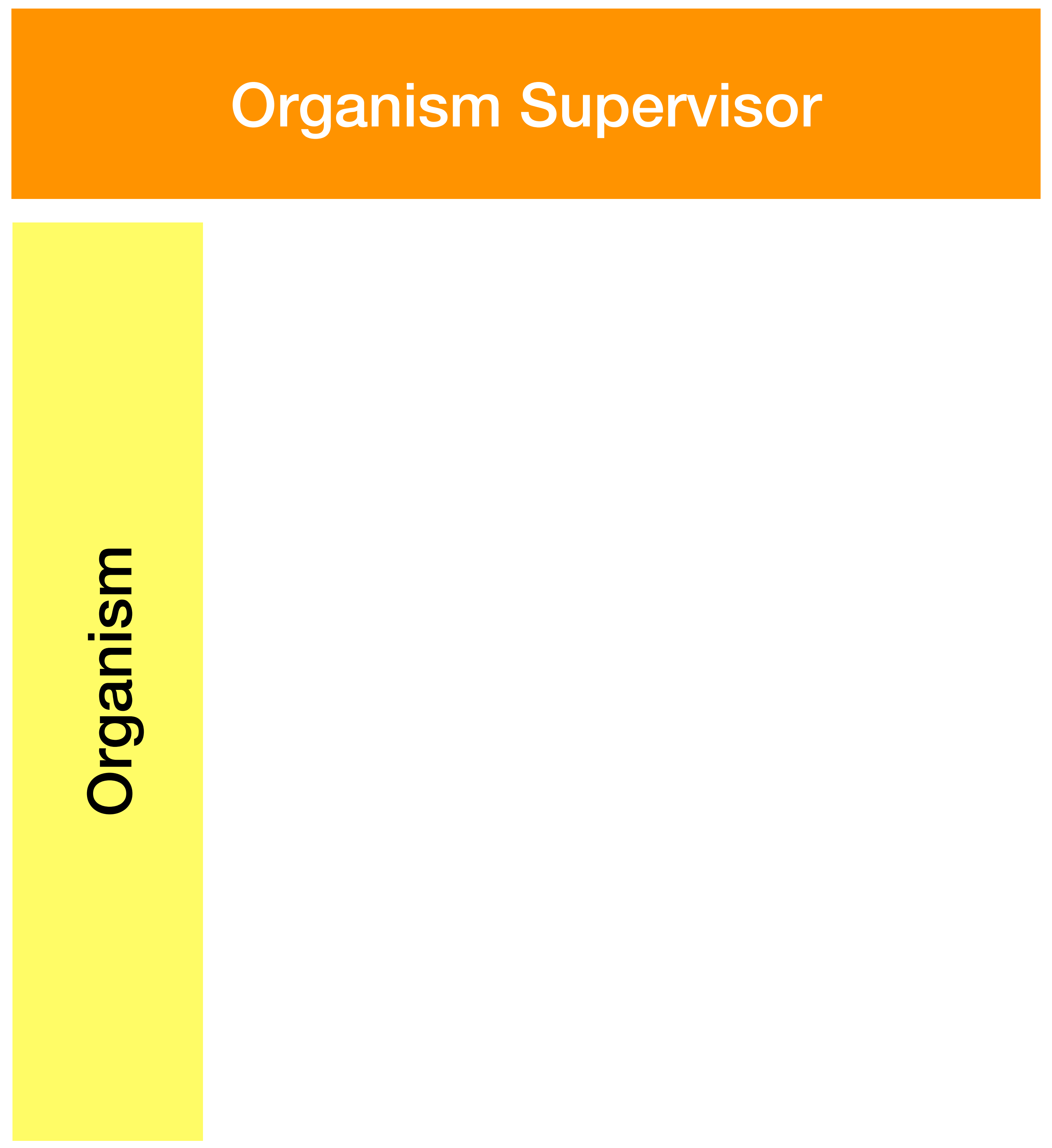
change_max_count/1

spawn/1

*spawn_organism/1

lib/prototype/organisms/organism_generator.ex

*Building the Playground

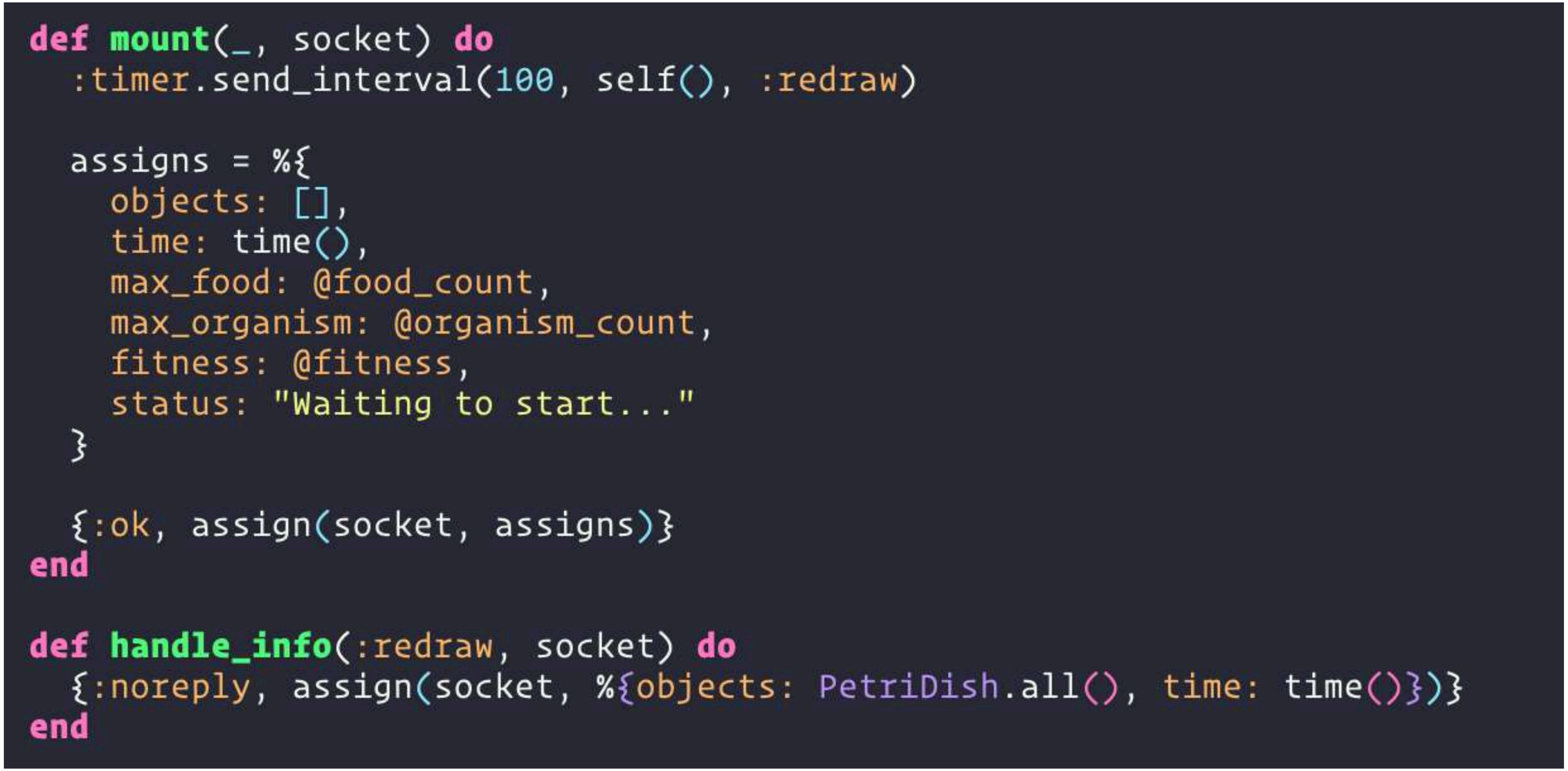


bounce/1

lib/prototype/organisms/organism.ex

*Didn't you say there's Phoenix LiveView?????

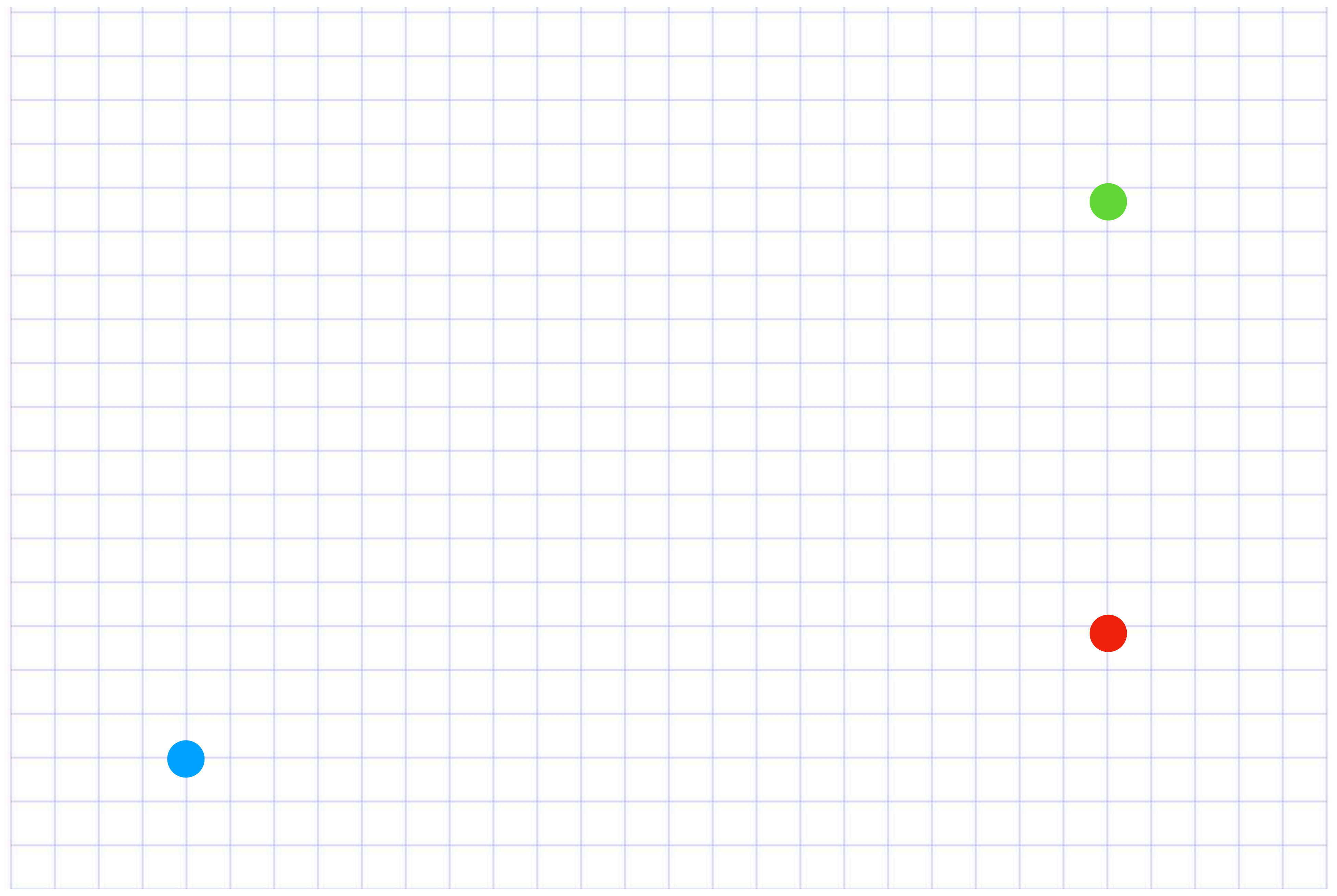
lib/prototype_web/live/petri_dish.ex



```
def mount(_, socket) do
  :timer.send_interval(100, self(), :redraw)

  assigns = %{
    objects: [],
    time: time(),
    max_food: @food_count,
    max_organism: @organism_count,
    fitness: @fitness,
    status: "Waiting to start..."
  }

  {:ok, assign(socket, assigns)}
end

def handle_info(:redraw, socket) do
  {:noreply, assign(socket, %{objects: PetriDish.all(), time: time()})}
end
```

*Only critical LiveView code

Hello, Little World! (one last time!)



Thank You!