Step 1 – download and open the netlogo #2 file from canvas

Please download model file from https://herts.instructure.com/courses/82224/modules/items/1486957 on canvas as this will be our starting point.

Step 2 – adding another type of agent (food) and more variables

Building on the existing code enter the additional lines to add food agents and more variables to the butterflies.

```
breed [ peoples people ]                    ; creating a population of people who will move around aimlessly
breed [ butterflies butterfly ]             ; creating a population of butterflies who will move around aimlessly but also seen the people
breed [ food flowers ]

butterflies-own [ people_seen people_hit    ; this creates 2 variables which will be used to count the total people seen and total people hit
health robustness speed_variation           ; this creates 3 variables for health, durability and speed
per_vis_rad per_vis_ang                     ; this creates variables for personalised vision cones
food_around_me closest_food                 ; this creates 2 variables to save the locations of food
]

food-own [ amount ]                         ; this creates a variable for the food to establish amount of the resource

globals [rad]                               ; this creates a global variable called rad
```

Step 3 – making our butterflies unique

To make our butterflies unique add the following lines to the create-butterflies section of code in the setup.

```
  create-butterflies number_of_butterfiles [; this creates the number of butterflies that your global variable states determined by the slider
    setxy random-xcor random-ycor           ; this sets the starting position of the butterflies to a random location in the world
    set color blue                          ; this sets the color of the butterflies to blue
    set size 10                             ; this sets the size of the butterflies to 10
    set shape "butterfly"                   ; this sets the shape of the butterflies to a butterfly

    ; making our butterflies unique
    set health 50 + random 50               ; this sets the health of the butterfly by adding 50 + a random alloctaion up to 50
    adjust_vision_cone                      ; this calls the adjust_vision_cone fuction to setup the vision cone
    set robustness random 10                ; this sets the robustness variable to a random value up to 10. lower means the butterfly is less a
    set speed_variation random 10           ; this sets the speed_variation variable to a random value up to 10. the higher the value the faste
    set heading 0                           ; this sets the starting heading of the butterfly to 0 (for demonstartion of speed difference)
    pen-down                                ; this puts the pen down so you can see where the butterfly moves
  ]
```

Step 3.1 – creating a vison cone function

After this create a function called adjust_vision_cone to customise the vision cone of each butterfly. The plan is to create a vision cone that is slightly random but also considers the health of the butterfly (I.e. if its not as healthy it cannot see as far or as wide). To do this enter the following code.

```
to adjust_vision_cone                                   ; this creates a function called adjust_vision_cone
  if ((vis_rad + random 20)*(health * 0.01)) > 0 [      ; if the calculation if greater than 0 then...
    set per_vis_rad ((vis_rad + random 20)*(health * 0.01))  ; set the personal vision radius to factor in some randomness and health
  ]
  if ((vis_ang + random 20)*(health * 0.01)) > 0 [      ; if the calculation if greater than 0 then...
    set per_vis_ang ((vis_ang + random 20)*(health * 0.01))  ; set the personal vision angle to factor in some randomness and health
  ]
end
```

Step 3.2 – updating the names of the vision cones in the rest of the code

Where there was previously a standard vision cone we now have a unique vision cone for each butterfly. The final part of implementing this new vision cone is to change the vis_rad and vis_ang to per_vis_rad and per_vis_ang wherever it has been used outside of the vision cone function. You could find these using CTRL F or look manually. They will be located where you are looking for people and in the visualisation sections of your code.

Step 3.3 – updating the speed section of the code for the butterflies

In the make_butterflies_move function modify the forward butterflies_speed code to the following:

```
    forward butterflies_speed + ( speed_variation * 0.1 ) ; moves butterfly forward by the butterflies_speed variable
```

* Now setup and run the model to see what happens

Step 4 – adding food to our model by calling a function

To add food agents to our world we will create a function that we can later use to continually add food to the environment. Write the following function below (not inside of!) the setup function.

```
to grow_food
  setxy random-xcor random-ycor             ; this sets the position of the food to a random location in the world
  set color yellow                          ; this sets the color of the food to yellow
  set size 10                               ; this sets the size of the food to 10
  set shape "plant"                         ; this sets the shape of the food to a plant
  set amount random 100                     ; this sets the amount of food per plant to a random value up to 100
end
```

Then add the following lines within the setup function.

```
  create-food 10 [                          ; this creates X number of new food plants for the butterflies to feed from
    grow_food                               ; this calls the grow_food function
  ]
```

* Now run the model and see what happens. You should have plants (food sources appear in your world).

Step 5 – continually growing food in the model

To continually add food to our model we can create a function to sprout new food with the following code.

```
to grow_more_food                       ; this creates a function called grow_more_food
  if ticks > 1000 [                     ; if the current number of ticks is greater than 100 then...
    ask patch random-xcor random-ycor [ ; ask a (1) patch in a random location (x, y coordinate) to do the following...
      sprout-food 1 [grow_food]         ; sprout (create new) food (1 in this instance) then call the grow_food function to set the parameters
    ]
    reset-ticks                         ; this resets the ticks counter back to 0
  ]
end
```

In order to enable this function add the following line to the go function.

```
  grow_more_food                        ; this calls the grow_more_food function
```

* now test the model and see what happens. Food should start growing in your world every 1000 ticks.

Step 6 – reorganising our code to make it easier to manage

Since our code is going to get a lot longer we want to make it easy to manage and to read. In order to do this we will break some of the existing code up into sub-functions as follows.

Step 6.1 – moving the visualisation aspects to a visualisation function

Cut and paste the visualisation code from the make_butterflies_move function into a new function called show_visualisations. The function should look as follows:

```
to show_visualisations                          ; this creates a function called show_visualisations
  if show_col_rad = true [                      ; this will switch on the visualisation of the collision radius if the switch is set to true
    ask patches in-radius rad [                 ; this sets up a radius around the butterfly to the value of the global variable rad which we
      set pcolor orange                         ; this sets the patch color to orange
    ]
  ]
  if show_vis_cone = true [                     ; this will switch on the visualisation of the vision cone if the switch is set to true
    ask patches in-cone per_vis_rad per_vis_ang [ ; this sets up a vision cone in front of the butterfly to the value of the global variables pe
      set pcolor red                            ; this sets the patch color to red
    ]
  ]
end
```

Now in place of this code in the make_butterflies_move function inside of the ask butterflies code to call the new function you have just created.

```
    show_visualisations                         ; call the show_visualisations function
```

* now test the model to check it still works

Step 6.2 – moving the people aspects to a separate function

Cut and paste the people aspects of the code into a new function called people function as per the instructions below. NOTE: this will be a return function so you will need to write **to-report** at the beginning of the function. The code should look SIMILAR to the following:

```
to-report people_function                          ; this creates a reporting function called people_function
  let seen [false]                                 ; this creates a local variable called seen
  let hit [false]                                  ; this creates a local variable called hit
  ask peoples in-cone per_vis_rad per_vis_ang [    ; this sets up a vison cone on the butterfly with the parameters from per_vis_rad
    set color green                                ; this sets the color of the person detected within the vision code of the butterf
    set seen true                                  ; this sets the local variable called seen to true indicating that a person has be
  ]

  ask peoples in-radius rad [                      ; this sets up a radius around the butterfly to the value of the global variable r
    set hit true                                   ; this sets the local variable called hit to true indicating that a person has col
  ]

  ifelse seen = true [                             ; if then else statement based on the local variable seen, if seen = true then...
    set people_seen people_seen + 1                ; add 1 to the people_seen count
    set color white                                ; set color of butterfly to white
    right 180     ;--------------------------------; set heading of the butterfly to 180 (turn around to avoid!)
  ][                                               ; if seen = false...
    right (random bwr - (bwr / 2)) ;---------------; this turns the butterfly right relative to its current heading by a random degre
  ]

  if hit = true [                                  ; if statement based on the local variable hit, if seen = true then...
    set people_hit people_hit + 1                  ; add 1 to the people_hit count
    set color green                                ; set color of butterfly to green
    set health health - robustness ;+++++++++++++++; adjust health of butterfly to health - collision penulty (robustness)
    adjust_vision_cone ;++++++++++++++++++++++++++; calls adjust_vision_cone to update the vision parameters based on heath changes
  ]
  report seen ;++++++++++++++++++++++++++++++++++; return true or false based in local variable seen
end
```

NOTE: you will eventually comment out the --- bits of code but leave in place for now. You should add the +++ bits of code as we will use these later.

Because this new function returns a binary value we need to create a local variable and call the newly created function to fill that variable. Enter the following line in place of the original code.

```
    let have_seen_person people_function               ; this creates a local variable called have_seen_person the fills it with the return
```

* now test the model to check it still works