Class';

Bee- Properties { double Health, boolean IsDead, double HealthLimit}

Health is the current health of the bee and will be initialised at 100.

IsDead is initialised as false.

I decided to add a health limit property to the class, so a quick comparison could be made against the health.

I used the double datatype so I could use the math truncate function as float does not work as an input parameter. Double also has the benefit of being able to store a larger number with more digits after the decimal point, which could come be useful if appropriate use cases arise in the future.

Methods {Damage(int damageDealt), CheckDead}

I created the damage method as follows:

There are three conditions that have to be met for damage to be dealt to a bee. (damageDealt variable had to be less than 100, damageDealt variable had to be greater than 0 and the bee had to be alive.)

After the if statement, I then subtracted from the bees current health the damageDealt variable divided by 100 multiplied by the current Health.

In practice I divided the current health by 100 as to make sure integer division was not used in the calculation.

The damage dealt method finishes by calling the CheckDead method.

The CheckDead method compares the HealthLimit variable to the current health of the bee, and if

the current health is less than the health limit IsDead becomes true.

CheckDead is private as it should only be accessed via the damage method.

The CheckDead method is separate from the damage method as each method should compute one thing.

Print Method was added to make presenting information onto a form easier and look nicer.

I used yes and no instead of true and false for Isdead and I thought that it would be easier for the average user to understand.

I decided health should only have two decimal places when being printed as that stopped rows from moving too much.

In each of their respective constructors the worker, queen and drone HealthLimits are set to 70,20 and 50 respectfully.

Reasons for each test (Reasons are given as comments in each Nunit test):

DamageMethodTest1 - This test is to check if the damage method works when health is at maximum health.

DamageMethodTest2 - This test is to check if the damage method works when health is not at maximum health.

DamageMethodTest3 - This test is to check if the damage method doesn't cause an error when damageDealt input is too low.

DamageMethodTest4 - This test is to check if the damage method doesn't cause an error when damageDealt input is too high.

DroneAliveTest - This test is to check if the drone is still alive after 10 damage.

DroneDeadTest - This test is to check if the drone is not still alive after 60 damage.

WorkerAliveTest - This test is to check if the worker is still alive after 10 damage.

WorkerDeadTest - This test is to check if the worker is not still alive after 40 damage.

QueenAliveTest - This test is to check if the queen is still alive after 10 damage.

QueenDeadTest - This test is to check if the queen is not still alive after 90 damage.

All tests currently pass.

The form methods:

changeLabelTexts, I created this method as I had repeated code for assigning the label texts in the button click method and the Form1 method. I chose a for loop instead of a foreach loop here as I am iterating through two lists at the same time.

initialiseBeeList and initialiseLabelList are two void methods that i used to make the Form1 method look neater.

button1\_click, I use a foreach loop to iterate through the bee list and call the damage method for each bee using a pseudo random number using the Random class.

For the interface I chose to separate each type of bee in columns so the data is easier to read.

I decided to just use a form as it quick to create and fits the user requirement of showing 30 bees and having a button.