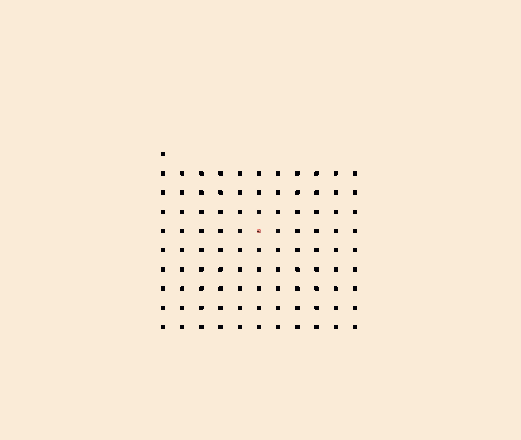
The goal of the course is to create a flocking mechanism for fish / birds.

Step 1

Examine the code, in particular examine functions which are marked as needing to be written by you.

Create and position 100 fish within the world.

* Use the *PlaceFish* function in **main.cpp**
* All fish must be visible
* Each fish must be separated from other fish
* Ideally all should be placed in a square, circle or spiral



Step 2

In DrawableGameObject createRandomDirection function,

* Set m\_direction to a random direction (or call *setDirection*) - **only on the x and y axes**
* **Whenever changing m\_direction it should always be normalised**
* Call the 'createRandomDirection' function at the end of the DrawableGameObject constructor
* In update function,
  + Set m\_position to a new position - the idea being the fish moves in the m\_direction vector over time
    - t is a delta time factor, it stores the time elapsed since the last frame, so if the program is running at 60 frames per second, it will be 1/60 (about 0.018)
    - new position = old position + direction \* t \* speed
    - Choose an appropriate constant speed value
* Run your code and test. You may find that fish eventually swim offscreen, we will fix this next.
* You will see in the main.cpp *render* function, *checkIsOnScreenAndFix* is a function called which partially determines the position of the fish.
  + You will write a test in the *checkIsOnScreenAndFix* function to determine whether the fish have gone off the left or right sides of the screen (x) or upper and lower parts (y)
  + You have two options, either:
    - Move the fish so it appears on the opposite side
    - or
    - Change its direction such that it swims back into the screen
  + Whichever you choose, m\_position should be altered so the fish is back on the screen, m\_direction will need to be changed in the second option
  + **The code you write should be where there are 'todo' comments**
* Run and test your code

Step 3

In the *update* function you will see a *nearbyDrawables* function called. You do not need to alter this.

This calculates the nearby fish determined by the NEARBY\_DISTANCE constant.

There are two ways to obtain a list of fish. Either all fish:

DrawableGameObject\* pFish = (drawList[index]);

or nearby fish:

DrawableGameObject\* pFish = (drawList[m\_nearbyDrawables[index]]);

To loop through all fish do this:

unsigned int index = 0;

while (index < drawList.size()) {

DrawableGameObject\* pFish = (drawList[index]);

}

To loop through all nearby fish do this:

unsigned int index = 0;

while (m\_nearbyDrawables[index] != UINT\_MAX) {

DrawableGameObject\* pFish = (drawList[m\_nearbyDrawables[index]]);

}

* Complete the *calculateCohesionVector* function. The idea is to return a vector (XMFLOAT3) which has a direction to the average **position** of all fish
  + In the *update* function, after calling this, update the fish direction to this vector
* Run your code. What happens? Do some / all fish swim to the centre?
* Complete the *calculateSeparationVector* function. You should return a vector which points away from the **position** of nearby fish
* In the update function, combine the cohesion and separation vectors (add and divide by two).
* Complete the *calculateAlignmentVector* function. You should return a vector which is the average **direction** of nearby fish.
* Combine this with cohesion and separation vectors.

Step 4

* Create weight constants for cohesion, separation and alignment, e.g. 1,1,3. These should weight the vectors before you combine them.
* Play with these weights, the number of fish (in main.cpp) and NEARBY\_DISTANCE and see if you can produce some interesting unexpected behaviour.
* NOTE: if you raise the number of fish too high, the program will run very slow, in this case you might find it beneficial (bene**fish**ial?) to run it in release mode.