

## FIT3094 AI for Gaming

### Practical sheet #6

#### Part A : Boid Class

1. Design (*not* implement!) a Boid class that employs your Vector class from Practical sheet #4. The Boid class should contain at least a constructor, destructor and debug (`print()` or `toString()`) method and the following:

- `canSee()` - return *true* if this boid can see a particular boid
- `tooClose()` - return *true* if this boid is too close to a particular boid
- `move()` - change this boid's position based on its current velocity
- `draw()` - draw this boid.

#### Hints:

1. You might like to pass a parameter to the following methods that is a pointer to an array containing all boids in the simulation so that the boid doing the calculation can determine its local neighbourhood using `canSee()` and `tooClose()`.
2. Since the following methods require you to change the velocity of a boid and the calculation of the alignment vector requires each boid to look at the velocity of its neighbours, it is important that you store the updated velocities and positions of the boids in temporary "future" velocity and position data members whilst you run through all the boids in the world. Then, when you have calculated new positions and velocities for *all* boids, you can set their "future" velocities and positions to be their "current" velocities and positions. This avoids allowing some boids to see into the future by mistakenly giving them access to the future state of their neighbours.
  - `calculateFlockCenteringVector()`  
Calculate a Vector from the current boid's location to the *local* (visible neighbours only) flock's center.
  - `calculateFleeVector()`  
Calculate a Vector from the current location away from any boids that are too close to this boid.
  - `calculateAlignVector()`  
Calculate a Vector in the average direction of travel of the visible neighbours of this boid.
  - `updateVelocity()`  
Determine this boid's *future* velocity by calculating flock centering, fleeing and alignment vectors.
  - `update()`  
Update the *current* velocity for this boid from the *future* velocity, then move this boid. (Can only do this safely after calling `updateVelocity()` for all boids - see Hint 2 above.)

## Part B : World Class

2. Design a World class. The class must contain an array of Boids. It will also need to contain a constructor, destructor, debug (`print()` or `toString()`) method and the following:

- `update()` - run through all of the boids in the World and update each one
- `draw()` - draw each of the boids in the World

## Part C : Assignment Programming Commencement

Once you have finished your design for Parts A and B above, implement and test your classes, starting with the Vector class from Practical sheet #4. If you have designed them properly, these classes should form a strong foundation for your Assignment Part 2 submission.

