# COMP1345

## INTRODUCTION TO APPLICATION DEVELOPMENT

**SEMESTER 02, 2014/15**

## PROJECT

## DETAILED REQUIREMENTS DOCUMENT

## Contents

# 1 INTRODUCTION

This document describes the detailed requirements for the final module project. The assignment consists of a series of tasks of varying complexity. All the tasks available can contribute to the development of the `PetAFish` simulation, a simple 2D fish simulation, as explained in the following sections. The development framework will be provided to allow the completion of the tasks.

***Hint 1: In the workshop in week 3 there was an exercise on randomized numbers, if you haven't completed that activity, it might be good to go back and take a look***

***Hint 2: Random Numbers aren't really random, they are seeded usually from system time. This means if two random number generators are created at exactly the same time, you will get exactly the same set of random numbers. An easy solution, create one, pass it as a parameter to everything that needs to use it and just call the .next() method from the objects that use it***

***Hint 3: Task 1 might take you the longest amount of time, but much of the logic and problem solving you will have to overcome in take 1 will help you to complete tasks 2,3 and 4, so don't panic if task 1 takes a long time!***

# 2 ASSESSMENT RATIONALE

In this project you are required to individually and should complete all elements of each task. The assessment 2 percentage, which is out of 50% of your total grade weightage is as follows:

- **Milestone 01** – Task Assessment + Learning Journal**, 15%**
- **Milestone 02**– Completing the tasks via code + Learning Journal **35%, break-down is as follows:**
    - Task 1 – **10%**
    - Task 02, Task 03, Task 04 – **15%**
    - Learning Journal – **10%**

In **task 02, 03, 04**, there is an **additional component**. The tasks generally involve some self-directed study or deep thinking.

For details on **submission requirements** and deadlines, please see the submission section at the **end of this document**.

In certain activities you will see **Standard Requirements** and **Implementation Requirements,** in this case the main marks will go for implementation requirements but you may get marks for **good code practices** inside even the standard requirements. Good code practices include:

- Appropriate Naming Conventions and Comments
- Working code based on suggested approaches
- Using appropriate methods, parameters, conditional statements and iterations

Each task has an associated maximum score. As a general rule, **you will not earn any credits for a given task if the code that you produce cannot be run and tested**, producing visible results coherent with the requirements. Hence, code that can be run and tested can earn you credits even if it has bugs or is incomplete, but writing new code that does not work at all will not be helpful!

For each task you will actually earn points depending on: i) how you fulfil the specific task requirements; ii) the quality of your solution, in relation to the assessment criteria for the task.

For all tasks in which you engage you will have to document the code that you produce appropriately. Comments must concisely clarify/justify your solutions. If you do not comment your code appropriately, you will not be able to earn the maximum score available. Plus, comments will be considered essential to demonstrate code ownership in case of suspected plagiarism.

# 3 RESOURCES

## 3.1 DEVELOPMENT FRAMEWORKS

All the project tasks rely on pre-existing code, provided through either the `FishORama` framework. The `FishORama` framework provides classes to implement simple 2D graphic simulations in an aquarium environment and is a MS Visual Studio 2010 /XNA solution, ready to be run and/or modified to implement your project tasks.

As described in more details in the following sections, the final project objectives will always require you to develop code for the `FishORama` framework.

## 3.2 HARDWARE AND SOFTWARE USER REQUIREMENTS

In case you decided to work on your own computers, you will have to take into account the following. The framework can be used through the Microsoft Visual Studio 2010 or the Microsoft Visual C# 2010 Express development environments. In addition to that, `FishORama` also requires the Microsoft XNA Game Studio 4.0 to be installed (although you will not use it directly). Details for running in on later versions of windows such as windows 08 and Visual Studios 2013 can be found on blackboard.

# 4 SUBMISSION

Submission requirements (i.e. what to submit, how and when) are specified for each task, in accordance with the assessment outline on blackboard

# 5 TASK 01: SUPAORANGEFISH

## 5.1 BRIEF

### 5.1.1 REQUIREMENTS

The company *Mak3ASym4U* decided to create `SupaOrangeFish` as the first modification to the `FishORama` simulation. The main objective here is to implement a number of seemingly intelligent behaviours in the basic Orange fish to give him a mind of his own and behaviours more natural to that of a fish inside an aquarium. Specifically, *Mak3ASym4U* have specified the following requirements:

1. **Dash Behaviour:** The orange fish ***randomly*** increases his speed by 10 (so if he was moving at a speed of 2, his new speed would be 12) from his current position to 250 pixels ahead of his current position, then returns to his natural speed. 250 pixels would relate to the distance travelled even if that distance includes the fish hitting the edge of the screen and turning back.
    a. **The behaviour of the fish moving from side to side and turning around when it hits the edge of the screen is naturally expected as a progression from class activities**
2. **Acceleration Behaviour:** A separate but similar behaviour to the dash, in this case the orange fish speed should accelerate and decelerate incrementally rather than switch directly to a new speed. This means, if the speed is 2 and we increase the speed to 12, there should be an incremental increase in speed to reach 12 rather than moving directly from a speed of 2 to a speed of 12. Once ***15 seconds*** have passed, the speed will **decrement progressively** to its original speed again.
3. **Hungry Behaviour:** At ***random intervals***, the fish will move back and forth 150 pixels on the X Axis while moving at a speed upwards of 1 on the Y axis, searching for food. This behaviour will last for 5 second before returning to his natural swim pattern. If the fish hits the top of the screen he should ***NOT*** start moving down again but hover at the top until the time it would take him to move 150 pixels.
4. **Sink Behaviour:** At ***random intervals***, the fish will swim down the screen by a random number of pixels between 50 and 150. You should ensure the fish does not swim off the bottom of the screen.
5. **Integration:** You must ensure that all of the above actions can be carried out without having an impact on any of the other actions. For example, if the fish is dashing, he should not be able to move into hungry behaviour or sink behaviour. Therefore only one behaviour should be carried out at any one time.

### 5.1.2 SPECIFIC ASSESSMENT CRITERIA

In this task you will be credited for:

1. Fully implemented and working code based on the requirements of each task.
2. Correct use of variables, placement of variables, scope and conditional statements.
3. Full integration of the tasks and efficiency of code so that everything runs with no conflict.
4. Clarity and relevance of code comments.
5. Use of naming conventions.

**IMPORTANT NOTE**: for implementation tasks, credits will be granted only based on at least partially working code, directly related to the task requirements and producing visible results on the screen (i.e. what doesn't work at all or doesn't produce any visible result will not be assessed).

# 6 TASK 02: SEAHORSE

## 6.1 BRIEF

### 6.1.1 REQUIREMENTS

The company *Mak3ASym4U* decided to create a new species for the `FishORama` called the `Seahorse`.***The standard requirements are minimal expected with the implementation being gradable***. The seahorse must move crossing the screen from side to side, back and forth, jumping as it progresses. The simulation token must be created based on the following specifications:

**Standard Requirements**

1. The token must be represented by a class called `SeahorseToken`. This class already exists in the framework but has not yet been fully implemented. You will be required to use any similarities between it and the `OrangeFishToken` you can find to implement it fully.
2. The token must be associated to a graphic asset labelled "`SeahorseVisuals`" and created and loaded in the library using the resource "`seahorse`", already integrated in the simulation (as it can be seen in the `FishORamaContent` project). Note: names are case sensitive.
3. The token must be created and placed in the aquarium through the `Kernel` class, similarly to what is done with the `OrangeFishToken` token.

**Implementation Requirements**

4. 3 seahorses should be created via the Kernel through a loop and stored inside an array where during creation they are assigned random x and y coordinates based on the viewable screen space. They should also be assigned a random speed between 5 and 10 which will determine their horizontal and vertical swim speed.
5. The SeahorseToken should be controlled via a `SeaHorseMind` for its movement, which should consist of moving up 100 pixels and down 100 pixels from a centre point while it moves back and forth across the screen (so if it starts at a Y position of 50, it would move up to 150 and down to -50)

### 6.1.2 ADVANCED ASSESSMENT CRITERIA

**IMPORTANT** : In this part it is expected that you would be required to do some external research beyond the scope of what has been covered during the lessons.

6. Add an additional behaviour that causes the sea horses to scatter in random directions once the chick leg has been placed on the screen. (note: the implementation for the token representing the chicken leg is provided with the framework; the chicken leg can be placed in the aquarium by left-clicking at the insertion position). The scatter should involve moving in a random direction away from the chicken leg at a higher speed than the Seahorse was initially moving for 100 pixels (ensuring of course the Seahorse remains within the screen border)

   **IMPORTANT NOTE**: for implementation tasks, credits will be granted only based on at least partially working code, directly related to the task requirements and producing visible results on the screen (i.e. what doesn't work at all or doesn't produce any visible result will not be assessed).

### 6.1.3 SPECIFIC ASSESSMENT CRITERIA

In this task you will be credited for:

1. Use and understanding of classes and objects.

2. Fully implemented and working code based on the requirements of each task.
3. Correct use of variables, placement of variables, scope and conditional statements.
4. Correct use of Methods and Parameters.
5. Correct use of Arrays in the appropriate classes
6. Full integration of the tasks and efficiency of code so that everything runs with no conflict.
7. Clarity and relevance of code comments.
8. Appropriate use of naming conventions.

# 7 TASK 03: PIRANHA

## 7.1 BRIEF

### 7.1.1 REQUIREMENTS

Based on the success of the Seahorse add on to the `FishORama` simulation, *Mak3ASym4U* have decided that adding a bit more spice to the fish tank might heat things up even more. ***The standard requirements are minimal expected with the implementation being gradable.*** The `Piranha` fish will be that spark that scares the Seahorse away as just the sight of a chicken leg wasn't considered scary enough to make a fish run away. The `Piranha` simulation token must be created based on the following specifications:

**Standard Requirements**

1. The token must be represented by a class called `PiranhaToken` controlled by a `PiranhaMind`
2. The token must be associated to a graphic asset labelled "`PiranhaVisuals`" and created and loaded in the library using the resource "`Piranha`", already integrated in the simulation (as it can be seen in the `FishORamaContent` project). Note: names are case sensitive.
3. The token must be created and placed in the aquarium through the `Kernel` class, similarly to what is done with the `OrangeFishToken` token.
4. The token should be placed on a random x,y location on the screen

**Implementation Requirements**

5. When the piranha is `Hungry`, the piranha will enact horizontal swim behaviour with a speed value of 5 (i.e. the horizontal position must be incremented by 5 units at each step. While swimming, the piranha will be constantly checking if there is a chicken leg in the aquarium (note: the implementation for the token representing the chicken leg is provided with the framework; the chicken leg can be placed in the aquarium by left-clicking at the insertion position).
6. As soon as the chicken leg appears, the piranha will switch from being `Hungry` to `Feeding`, reaching the food and eating it. The piranha will have to make the chicken leg disappear (note: to make the object disappear it is possible to use the method `RemoveChickenLeg` contained in class `Aquarium`).
7. After eating a chicken leg, the Piranha will return to his natural `Hungry` swimming behaviour again.

### 7.1.2 ADVANCED ASSESSMENT CRITERIA

**IMPORTANT** : In this part it is expected that you would be required to do some external research beyond the scope of what has been covered during the lessons.

1. Have the Piranha return to its original patrol position once the chicken leg has been removed and to represent his `Full` state it should slow to a speed of one for 5 seconds before returning to its natural speed.

### 7.1.3 SPECIFIC ASSESSMENT CRITERIA

In this task you will be credited for:

2. Use and understanding of classes and objects.
3. Fully implemented and working code based on the requirements of each task.
4. Correct use of variables, placement of variables, scope and conditional statements.
5. Correct use of Methods and Parameters.
6. Full integration of the tasks and efficiency of code so that everything runs with no conflict.
7. Clarity and relevance of code comments.
8. Appropriate use of naming conventions.

**IMPORTANT NOTE**: for implementation tasks, credits will be granted only based on at least partially working code, directly related to the task requirements and producing visible results on the screen (i.e. what doesn't work at all or doesn't produce any visible result will not be assessed).

# 8 TASK 04: BUBBLES

## 8.1 BRIEF

### 8.1.1 REQUIREMENTS

Although the fish are behaving very well indeed, *Mak3ASym4U* have decided that the OrangeFish just isn't as exciting as the newly create Piranha and Seahorse additions. You have been tasked with implementing a fish based bubble system into the `FishORama` simulation. The bubbles must be handled in an efficient manner as due to the current economic climate *Mak3ASym4U* can only afford 10 bubbles. ***The standard requirements are minimal expected with the implementation being gradable.*** To achieve this goal, you are expected to fulfil the following requirements:

**Standard Requirements**

1. The token must be represented by a class called `BubbleToken`.
2. The token must be associated to a graphic asset labelled "`BubbleVisual`" and created and loaded in the library using the resource "`Bubble`", already integrated in the simulation (as it can be seen in the `FishORamaContent` project).
3. The behaviour of the token must be controlled by a mind represented by a new class, called `BubbleMind`.
4. Create 5 `BubbleToken` stored inside an array and ensure each one is placed in a layer higher than any of the fish inside the tank (meaning each bubble if passing a fish should pass in over the fish and not behind it) For this implementation, there is ***no need*** to create a `BubbleMind`, but if you would prefer to do that it is acceptable

**Implementation Requirements**

5. The initial position of each bubble should be in the same position as the fish and placed on the scene by using a loop and array to enact mScene.Place.
6. Randomize the speed of each bubble between 3 and 5 so that different bubbles will move at different speeds.

7. When a bubble reaches 150 pixels past the position it started from, reposition it at the fishes current position and randomize the speed again.
   ***Hint: You can give the OrangeFishMind access to the bubble array by creating a public data member (or using a property) inside the OrangeFishMind, and setting it to the bubble array from Kernel, this way the mind can have access to all created bubbles and Update their positions or call their methods(even access their public data members or properties). Not the best solution but perhaps the most straight forward. More advanced solutions would have the bubble responsible for moving itself and finding the position of the OrangeFishToken, but the easier solution will be perfectly fine in this case.***

### 8.1.2 ADVANCED ASSESSMENT CRITERIA

**IMPORTANT** : In this part it is expected that you would be required to do some external research beyond the scope of what has been covered during the lessons.

1. Using the features of the `Math` library, implement the simple use of a `sin` or `cos` wave to allow the bubble to move in a nice swaying motion as it moves up the screen.

### 8.1.3 SPECIFIC ASSESSMENT CRITERIA

In this task you will be credited for:

1. Use and understanding of classes and objects.
2. Fully implemented and working code based on the requirements of each task.
3. Correct use of variables, placement of variables, scope and conditional statements.
4. Correct use of Methods and Parameters.
5. Appropriate use of Arrays and Loops.
6. Full integration of the tasks and efficiency of code so that everything runs with no conflict.
7. Clarity and relevance of code comments.
8. Use of naming conventions.

**IMPORTANT NOTE**: for implementation tasks, credits will be granted only based on at least partially working code, directly related to the task requirements and producing visible results on the screen (i.e. what doesn't work at all or doesn't produce any visible result will not be assessed).

# 9 SUBMISSION

## 9.1.1 MILESTONE 01 (19/03/2014)

#### *9.1.1.1 DELIVERABLES*

You will be given a template document that must be completed to help you analyse the tasks given to you in terms of their complexity, problems you might encounter based on this complexity and a weightage as to the overall difficulty of the tasks. This should help you evaluate yourself, your time and the tasks.

#### *9.1.1.2 SUBMISSION INSTRUCTIONS*

The instructions below must be strictly followed in the submission process.

All your milestone 01 deliverables must be packaged in a single `.zip` file named after the following convention:

*{Your Student Number}*_COMP1345_2015_Milestone01.

The package should include an MS Word Document containing the completed task assessment outline based on the template provided.

Your project Milestone 01 package should then be uploaded on Blackboard, in the digital dropbox, accessible from: 'Assessment' > 'Assessment 2' > 'Milestone 01'.

Your submission will be accompanied by a ***learning journal*** a week after this milestone. All journals can be found on Blackboard under Assessment>Assessment 2>Learning Journals

### 9.1.2 MILESTONE 02 (07/05/2015)

#### *9.1.2.1 DELIVERABLES*

For milestone 02 you are required to submit a fully-functional release of the simulation, with all the requirements implemented.

#### *9.1.2.2 SUBMISSION INSTRUCTIONS*

The instructions below must be strictly followed in the submission process.

All your milestone 02 deliverables must be packaged in a single `.zip` file named after the following convention:

*{Your Student Number}*_COMP1345_2015_Milestone02.

The package should include an MS Visual Studio 2010 project for your prototype. The project must be called `FishORama_v1`, and must be ready to be extracted from the zip package and executed from Visual Studio 2010.

Your project Milestone 02 package should then be uploaded on Blackboard, in the digital dropbox, accessible from: 'Assessment' > 'Assessment 2' > 'Milestone 02'.

Your submission will be accompanied by a ***learning journal*** a week after this milestone. All journals can be found on Blackboard under Assessment 2->Learning Journals