

INFT3960 – Game Production

Week 05

Module 5.2

Object Oriented

Course Overview

Lec	Date	Modules	Assignments
1	Tuesday 30 Jul	Mod 1.1, Mod 1.2	
2	Tuesday 5 Aug	Mod 2.1, Mod 2.2, Mod 2.3, Mod 2.4	
3	Tuesday 12 Aug	Mod 3.1, Mod 3.2, Mod 3.3	★ Assign 1 12 Aug, 11:00 pm
4	Tuesday 19 Aug	Mod 4.1, Mod 4.2	
5	Tuesday 26 Aug	Mod 5.1, Mod 5.2	
6	Tuesday 3 Sep	Mod 6.1, Mod 6.2, Mod 6.3	
7	Tuesday 10 Sep	Mod 7.1, Mod 7.2	★ Assign 2 12 Sep, 11:00 pm
8	Tuesday 17 Sep	Mod 8.1	
9	Tuesday 24 Sep	Mod 9.1	
10	Tuesday 15 Oct	Mod 10.1, Mod 10.2	
11	Tuesday 22 Oct	Mod 11.1, Mod 11.2	★ Assign 3 24 Oct, 11:00pm
12	Tuesday 29 Oct	Mod 12.1, Mod 12.2	

Course Details

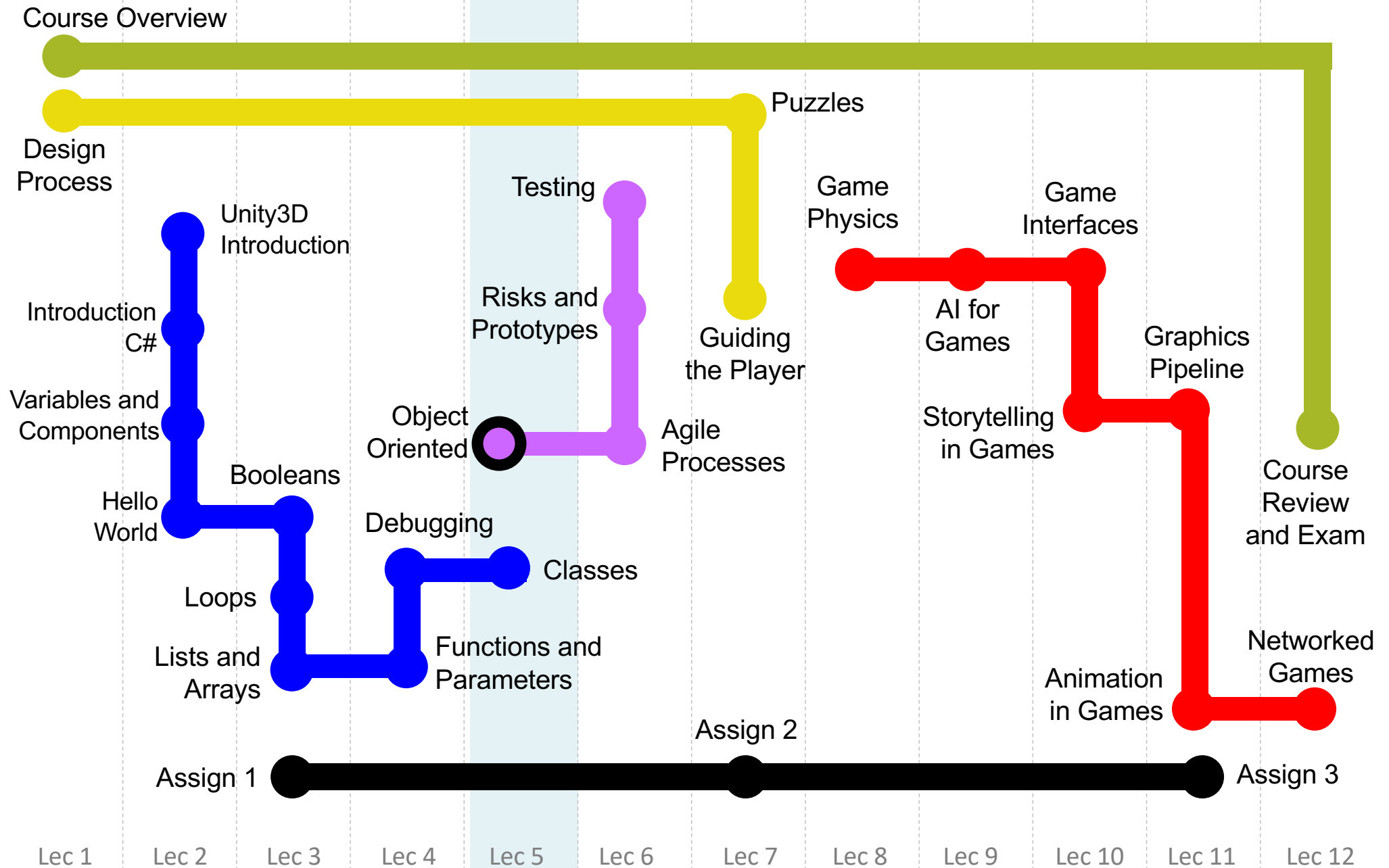
Game Design

Unity 3D and C#

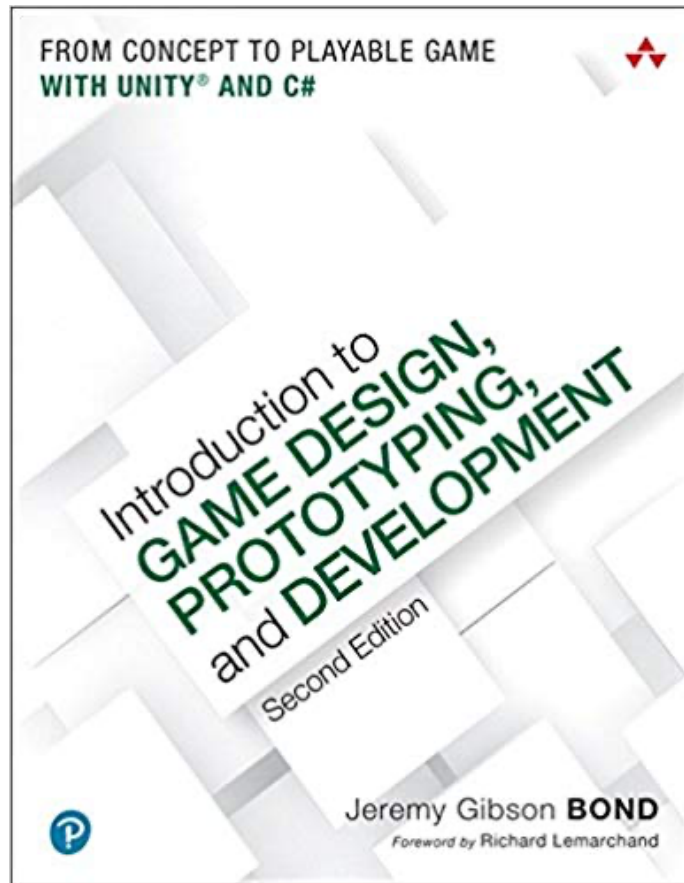
Development Process

Core Game Concepts

Assignments




Object Oriented – (Chapter 27)



OBJECT
ORIENTED
THINKING

Object Oriented – Topics

- 
- The Object-Oriented Metaphor
 - Object-Oriented Flocks of Birds
 - Modularity

Object-Oriented Metaphor

Consider Object-Oriented Programming as it relates to a flock of birds

For hundreds of years, the motions of bird flocks (and schools of fish) fascinated researchers

- How do hundreds of individuals move in such coordinated behavior?
- Which animal "leads" the flock? How is the leader chosen?

Object-Oriented Metaphor

To simulate a flock on a computer in a monolithic way, it was thought that a massive function would be required

1. Track every bird's position
2. Make global decisions about the behavior of the flock
3. Figure out how those global decisions would affect each bird

But in a real flock, each bird looks at her own environment, goals, etc. and makes decisions

- Each bird acts as an individual based on her understanding of the world
- This is the core concept of Object-Oriented Programming (OOP)

Flocks of Birds

BOIDS provides another approach to flocks

"Flocks, Herds, and Schools: A Distributed Behavioral Model" (1987) by Craig W. Reynolds

A simple, object-oriented approach to flocking behavior that Reynolds called "Boids"

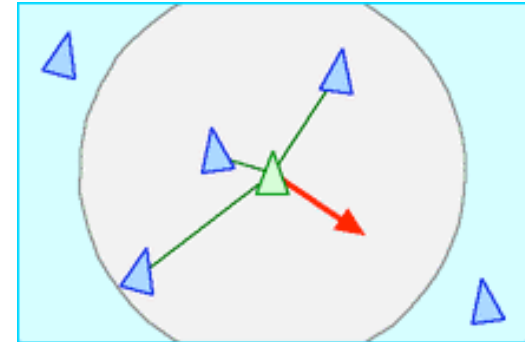
Boids uses three simple rules:

- Separation
- Alignment
- Cohesion

Boids – Three rules

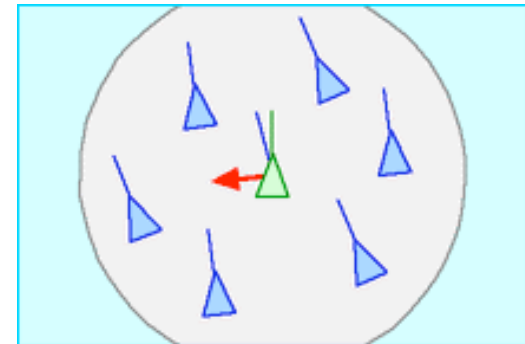
Separation

Avoid crowding nearby flockmates



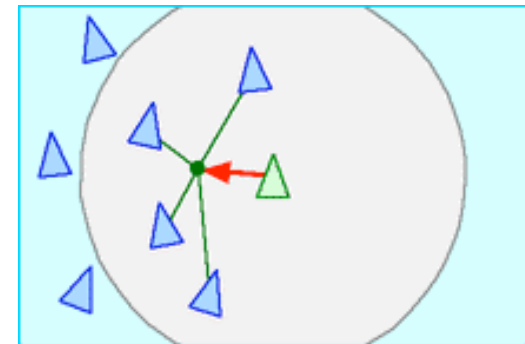
Alignment

Match heading with nearby flockmates



Cohesion

Try to center self relative to nearby flockmates

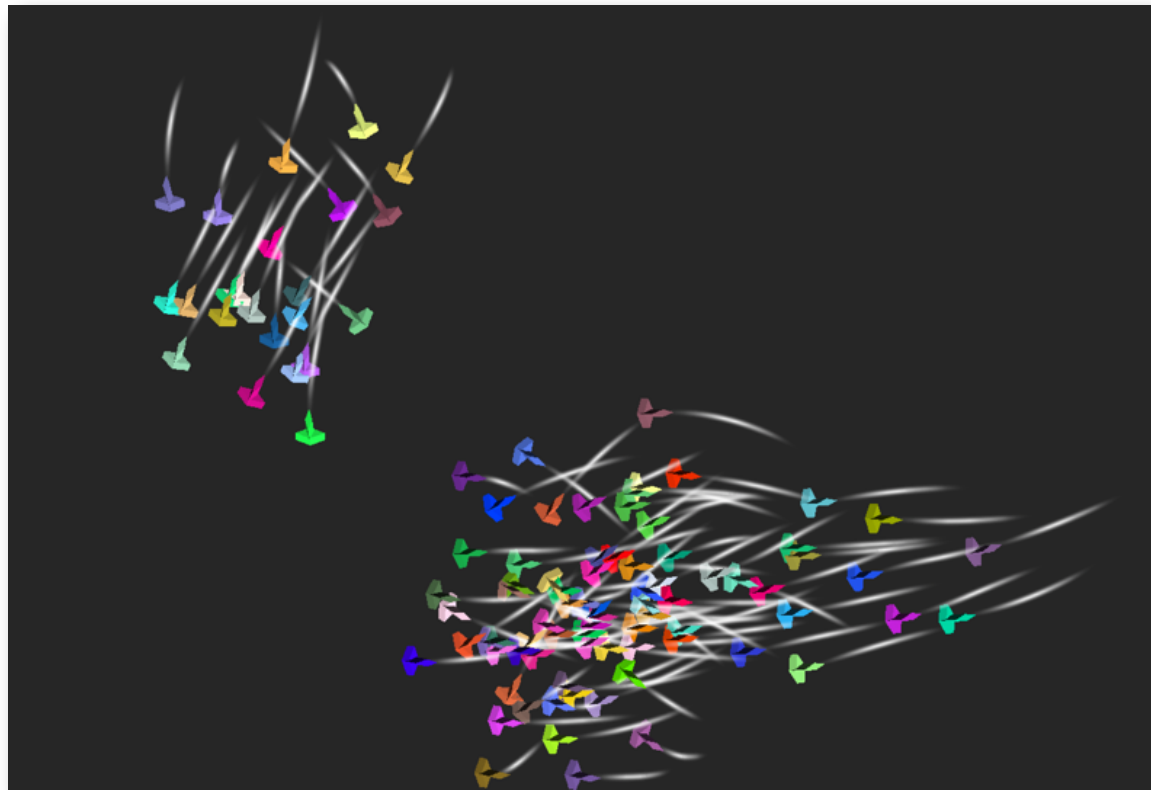


Boids – Three rules

These three rules create surprisingly good flocking behavior:

<https://www.youtube.com/watch?v=86iQiV3-3IA>

In this chapter of the book, you create a simple 2D version of Boids that runs in Unity



Modularity

Another aspect of OOP is Modularity

Through modularity, each piece of code is expected to follow specific parameters

- But the internal implementation of that code is up to the individual developer

This enables teams to effectively collaborate on code

- Each programmer receives a spec for what her part of the code is supposed to do
- That code is encapsulated into one or more classes which are clearly documented

Modularity

Collaboration benefits of modular code:

- The public fields and methods of each class are pre-specified
- Other programmers can write code that will interact with each class without knowing the internal implementation of that class
- The internals of the class can be replaced without affecting any other code

However, top-down modularity is often difficult to define ahead of time in game prototypes

- Because the spec for a prototype changes rapidly
- Instead of top-down, just think bottom-up about making your code easily reusable in later projects
- Make each reusable element of your code a module

Summary

In OOP, each instance of a class thinks for itself -
Rather than all being controlled by a single function

Through OOP, simple rules can be embedded into
each class member

From these simple rules can emerge complex,
interesting behaviors

Modular code can help programming teams
collaborate