

GDD 120 – Introductory Programming for Game Developers - 3 Credit Hours Spring 2009

COURSE MATERIALS

Textbooks: There are no textbooks for this course. Instead, we'll be using the textbook that I develop as we go along.

Class Handouts

iClicker

eCompanion: <http://www.uccs.edu/~online/login2.htm>

COURSE OBJECTIVE AND DESCRIPTION

The objective for this course is to learn the key foundational concepts associated with using computers to solve problems using Object-Oriented Programming with C# as the programming language – oh yeah, and to have fun! Of course, we'll be solving problems in the game development domain, but the concepts you learn in this class will also be applicable to any problem you might try to solve with a computer. Developing game software is a very fun thing to do; this course will start you on the road to doing just that.

The specific course outcomes that the Computer Science Department has identified for the corresponding course for computer science (CS 115) are provided at the end of the syllabus. Most of them won't mean anything to you at this point, but by the end of the course you should be able to achieve most or all of the listed outcomes (though of course in C#, not Java).

COURSE SCHEDULE

| Lesson | Date | Reading | Topics | Assignment Due |
|--------|------|--------------|--|--|
| 1 | 1/21 | None | Course Introduction, Overview of XNA and the IDE | |
| 2 | 1/26 | Chapters 2-3 | Your First C# Program | |
| 3 | 1/28 | Chapter 4 | Object-Oriented Design and Programming | |
| 4 | 2/2 | Chapter 5 | XNA Basics | Programming Assignment 1 |
| 5 | 2/4 | None | Lab Work | Lab Assignment 1 |
| 6 | 2/9 | Chapter 6 | Unit Testing | Programming Assignment 2 |
| 7 | 2/11 | Chapter 7 | Primitive Data Types, Variables, and Constants | |
| 8 | 2/16 | Chapter 8 | Strings | |
| 9 | 2/18 | None | Lab Work | Programming Assignment 3 Lab Assignment 2 |
| 10 | 2/23 | Chapter 9 | Selection | |
| 11 | 2/25 | Chapter 10 | XNA Input | Programming Assignment 4 |
| 12 | 3/2 | None | Lab Practicum 1 | |
| 13 | 3/4 | Chapter 11 | Arrays and Collection Classes | |
| 14 | 3/9 | Chapter 12 | Iteration: For and Foreach loops | Programming Assignment 5 |
| 15 | 3/11 | None | Mid-Term Exam | |
| 16 | 3/16 | Chapter 12 | Iteration: While loops | |
| 17 | 3/18 | Chapters 13 | XNA Audio | Programming Assignment 6 |
| X | 3/23 | None | No Class: Spring Break | |
| X | 3/25 | None | No Class: Spring Break | |
| 18 | 3/30 | Chapter 14 | Class Design, Part 1 | |
| 19 | 4/1 | Chapter 15 | Class Design, Part 2 | |
| 20 | 4/6 | Chapter 16 | XNA Text Output | |
| 21 | 4/8 | None | Lab Work | Programming Assignment 7 Lab Assignment 3 |

| | | | | |
|----|------|------------|------------------------------|---------------------------|
| 22 | 4/13 | None | Lab Practicum 2 | |
| 23 | 4/15 | Chapter 17 | Equality and Hash Codes | Programming Assignment 8 |
| 24 | 4/20 | Chapter 18 | Inheritance and Polymorphism | |
| 25 | 4/22 | Chapter 19 | Interfaces and Polymorphism | |
| 26 | 4/27 | Chapter 20 | Exceptions | Programming Assignment 9 |
| 27 | 4/29 | None | Lab Practicum 3 | |
| 28 | 5/4 | Chapter 21 | File IO | |
| 29 | 5/6 | Chapter 22 | XNA Controllers | Programming Assignment 10 |
| 30 | 5/11 | Chapter 23 | Course Wrap-up | |
| X | 5/18 | None | Final Exam (5/18) | |

GRADING POLICY

- You'll attend class and participate in each class.
- You'll complete 10 programming assignments.
- You'll complete 3 lab assignments.
- You'll complete 3 lab practica (in-lab programming tests)
- You'll take a mid-term exam and a final exam.
- Final grades are computed using the following weights:

| | |
|--------------------------------|-----|
| Class Attendance/Participation | 10% |
| Programming Assignments | 20% |
| Laboratory Assignments | 12% |
| Lab Practica | 21% |
| Mid-Term Exam | 13% |
| Final Exam | 24% |

- All grades are based on a scale from 0-100 as follows:

| | | |
|--------|---|---|
| 90-100 | = | A |
| 80-89 | = | B |
| 70-79 | = | C |
| 60-69 | = | D |
| < 60 | = | F |

A linear shift may be applied to final grade averages as a one-time scale at the professor's discretion.

TIME COMMITMENT

This class is going to be a lot of work for most of you. Between the readings and the programming assignments (discussed more below), you can probably plan on about 3 or 4 hours a week outside of class.

Can you successfully finish the class without putting in the time? Maybe, depending on your past programming experience and how quickly you get new concepts. It would certainly be reasonable, though, to at least plan on the projected time above until you find it takes you less (or more) time to complete these course activities.

PROGRAMMING ASSIGNMENTS

So here's the thing. Learning to program requires that you actually DO lots of programming – not just read about it, hear about it, or watch someone else do it, but actually do it yourself. The point of the programming assignments is to get you that practice that's so vital to your learning.

But programming takes time, right? Doesn't this mean you'll have to spend a lot of time working on this class? The short answer is yes, it does. Depending on your past programming experience, each of the programming assignments should take you 1-2 hours to complete. That means you should plan your time carefully.

For the programming assignments, you can work together in groups if you want. Although you definitely need to make sure you understand how everything works, there's certainly value in working with other people to try to learn new ideas. So, while you do need to spend time doing the assignments, you don't have to feel like you're alone in the wilderness doing them either. Work together as much as you want on this part of the course.

To get credit for a particular programming assignment, you must turn it in by the beginning of class on the given day; **no late submissions will be accepted**. Your turn-in must work, be fully commented, meet all our standards for style and documentation, and where required, include a test class that thoroughly unit tests your classes using NUnit. To turn in your assignment, zip up your entire project folder into a file named <yourlastname>.zip. **Note: naming your zip file something other than <yourlastname>.zip will automatically lose you 1 point; don't do it!** Log into eCompanion and submit the file into the appropriate dropbox.

Your grade will simply be a 2, 1 or 0, with no written feedback other than your grade. Programming assignments will be posted on eCompanion no later than one week prior to the due date for that assignment.

CLASS PREPARATION

It may be too early in your college-level academic career for you to realize this, but having a professor simply tell you what you could have read in the book is boring! Believe it or not, it's boring for professors too; we'd much rather build on the readings to help you really understand the course concepts. It's therefore much better for both of us if you do the assigned reading BEFORE class.

I know, I know – every teacher you've ever had has told you to do the reading before class. It probably just sounds like white noise to you now, and you figure if you wait through the soap-box speech you can just ignore the readings and figure I'll cover them in class. Although I'll certainly cover important concepts in more depth and answer questions from the readings, I plan to build on them, not parrot them back to you. That means I expect you to do the readings before class, not after or (egads!) during class.

CLASS ATTENDANCE AND PARTICIPATION

Learning how to program is hard for some people. Although in some ways learning a programming language is like learning a foreign language like French or Spanish, in most ways this is a different process. Not only do you need to learn the syntax (punctuation) and the semantics (meaning) of the programming language, you also have to actually figure out how to solve problems using the language. That's really the hardest part of all, and it's very unusual for a new programmer to pick up the necessary skills with just a book.

Luckily, I'm here to help! Lectures will be designed to build on the readings as described above while also helping you hone your programming skills. To benefit from this help, though, you need to attend class and you need to participate during class. Here's how I'll make sure that's actually happening.

Each student in the class is required to purchase an i>clicker student response device from the bookstore. During each class, I'll be using a number of questions to make sure you're getting key concepts from that day and to foster discussion. You'll use your i>clicker to answer those questions to achieve the above goals, but I'll also be able to use your responses to track your attendance and participation. If you're not in class, or you don't answer the questions, you won't get credit for participating on that day. Note that these questions aren't "mini-tests" because you don't have to answer correctly to get credit, but you do have to answer.

Class attendance/participation as measured through i>clicker use counts for 10% of your grade. Even if you have an excused absence from class (we'll work excused absences on a case-by-case basis), you are 100% responsible for all material and announcements covered in class.

PAIR PROGRAMMING

There's lots of evidence in industry that programming in pairs (called, surprisingly enough, pair programming) leads to higher-quality software. In addition, numerous professors have found that pair programming can lead to better student learning. The Computer Science Department regularly uses pair programming in our CS 115 lab assignments, and it seems to work pretty well. We'll therefore use pair programming in GDD 120 also. For each lab assignment, you'll work in randomly-assigned pairs to complete the assignment.

LAB PRACTICA

So with all this working together on programming assignments (if you want to) and on the lab assignments, how will I make sure each of you knows how to program individually? That's where the lab practica come in.

A lab practicum is a test that you'll take at a computer in the lab. Basically, I'll give you a problem to solve and you have to develop and test a program to solve that problem before the class period ends. If you've learned how to program as you go through the course, you'll be fine. If, however, you're riding on the coat-tails of others as you complete the programming and lab assignments, this is where you'll pay the price. Classes are about the learning more than anything else, and this is a way to make sure each of you is learning to program on your own.

CHEATING ON LAB PRACTICA AND EXAMS

Absolutely no cheating on the lab practica or exams will be tolerated; each student is expected to develop their own lab practicum and exam solutions.

Cheating will result in an AUTOMATIC 0 for the entire lab practicum or exam in question. For further details on academic honesty the student is referred to the University Catalog.

LATE DROP

Dropping of a class after the deadline listed in the class schedule is governed by departmental and college policy. The student must show documented evidence supporting reasons for a request to drop a class after the deadline. Each request is considered on an individual basis for determining acceptance.

OFFICE HOURS

I'll be available during my official office hours (posted on my web page) and by appointment. If you drop by my office outside my official office hours without an appointment, I may have time to see you, but I'll also feel free to have you schedule an appointment for a later time instead.

You can also contact me by e-mail with questions; I'll answer those questions as soon as I can, but I'll only be checking my e-mail during the day on weekdays. Note: I will NOT answer any email questions about assignments in the 24 hours preceding the assignment due date/time.

CS 115 COURSE OUTCOMES

- | |
|---|
| <ul style="list-style-type: none">• Understands the basics of computer problem solving and how implemented in the chosen language (currently Java).<ul style="list-style-type: none">○ Understands that all computer problem solving has two parts: data and operations.○ Understands that data must be declared (type and identifier) and initialized before use. Understands the difference between scalar and reference types.○ Understands the mechanics of operation definition and use.○ Understands the mechanics of installation and use of the chosen language (currently this means installation and setup of Java on a Windows based PC plus editing, compiling and running Java programs with and without the use of an IDE such as JBuilder). |
| <ul style="list-style-type: none">• Is capable of solving problems using fundamental types and operations.<ul style="list-style-type: none">○ String and integer types, newly defined types as new classes.○ Understands conditional, iterative and branching control constructs.○ Understands how to do simple input/output. |
| <ul style="list-style-type: none">• Understands the basic principles of object-orientation and application to problem solving.<ul style="list-style-type: none">○ Understands the defining OO concepts: Object, Class, Instance, Message, Method. Understands how to define and use classes and objects.○ Understands the essential OO features: abstraction, encapsulation, inheritance, polymorphism.○ Understands that OO programming consists of sending messages to objects (objects are data, messages invoke operations defined by methods).○ Understands that static fields and operations are rarely justifiable. Understands the meanings of modifiers for class, field and method.○ Understands the operations defined for all objects (in class Object) and the need to redefine selected inherited operations. |
| <ul style="list-style-type: none">• Understands selected advanced concepts in OO and Java.<ul style="list-style-type: none">○ Understands definition/rules/purpose/use of an interface and that it is also a type representing a precise but common behavior or property.○ Understands simple data structures (arrays and ArrayList) and how to create/use them.○ Understands the concept of an Iterator and how to use it to access objects in data structures.○ Has experience building/using classes in an application requiring the use of polymorphic substitution and dynamic binding for both sub-classes (inheritance) and classes that implement interfaces. |