CMPS 20: Game Design Experience

Course Overview
Introduction to XNA
Introduction to C#
Administrivia

• If you did not sign for CMPS 20, here’s your chance to leave

• Permission codes: Class is already overflowing so chances are slim for any more students to join in

• 80K – Game Design Fundamentals: Covers history and design of games; could become a major requirement; very useful and highly recommended
Administrivia

• CMPS 20
• Professor: Arnav Jhala (<a>jhala@cs.ucsc.edu</a>)
  – Office Hours: Monday 3 to 4:30 and by appointment
• Teaching Assistant: David Seagal (<a>drseagal@ucsc.edu</a>)
• Readers/Tutors: Robert Kavert, Adrien Young, Slade Villena

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Class Information

• Website
www.soe.ucsc.edu/classes/cmps020/Winter10

• Schedule (Lecture slides, notes, due dates)
• Homework and Project Information
  – Description and Evaluation Criteria
• Resources (Links to articles, tutorials, examples, etc.)
• Twitter: CMPS20W10

• Keep up with class

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Introductions

• Professor
• Teaching Assistant
• Students
  – Name
  – Major
  – One Favorite Game

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Course Intro: Objectives

• Learn basic principles of game programming
  – Main game loop, display of 2D sprites and 3D objects
  – Content pipeline, Art Integration
  – Collision detection, scrolling game worlds, shaders
  – Audio
• Learn basic game AI techniques
  – Simple behaviors, A* pathfinding
• Learn basic principles of object-oriented design
  – Subdividing a project into classes
  – Unified Modeling Language structure diagrams
  – Software design patterns
• Develop increased proficiency in programming
  – C# language, coding focused assignments
• Learn techniques for working as a team
  – Quarter-long game project developed in 4 person team
Grades

- Homework: 30% (3 assignments, each worth 10%)
- Midterm exam: 15%
- Final exam: 15%
- Term project: 40%, broken down as follows
  - (Percentages are of final course grade, and sum to 40%)
  - Team selection: 1%
  - Game concept document: 5%
  - Work breakdown and schedule: 3%
  - Technical design document: 7%
  - Partially operational game prototype: 3%
  - Updated schedule: 1%
  - Final game project: 20%
Reading Material

• Textbooks
  – Learning XNA 3.0 by Aaron Reed, O’Reilly publishers, 2008
  – Programming C# 3.0 by Jesse Liberty and Donald Xie, O’Reilly publishers, 2007
  – Available at campus bookstore and online

• Reference Materials
  – Articles that are uploaded on class website
  – Links to XNA and C# development forums, tutorials, etc.
Project

• Work in teams of 4 to create a fully playable computer game
  – Developed on XNA platform in C# (covered in class)
  – XNA provides libraries and art content (meshes, textures, etc.) is freely available online
  – Created games can run on Xbox 360, PC, and Zune
  – Examples

• Phases
  – Team Formation – Week 2
  – Game Concept Document – Week 4
  – Production Schedule Document – Week 5
  – Technical Design Document (including prototypes) – Week 7
  – Playable Game Milestone 1 – Week 8
  – Playable Game Milestone 2 – Week 9
  – Final Game – Week 10
XNA Game Examples

• Some of these were made in 48 hours over a weekend by groups of 3 to 4 programmers
• Student games
• Research projects
XNA Game Studio Express

- XNA GSE is a series of libraries for creating 2D and 3D computer games
  - Uses C# as the primary programming language
  - Integrated with Visual Studio C# Express
    - Also now the full version of Visual Studio
  - Games can run under Windows or on Xbox 360
  - It is possible to create professional games using this toolkit
- Example games:
XNA Game Studio Architecture

• You write your game in C#
  – Using features in XNA Framework
• Runs on top of common language runtime ("Managed Code")

<table>
<thead>
<tr>
<th>You provide</th>
<th>Provided for you</th>
</tr>
</thead>
</table>

- Game code (C#) & content
- XNA Framework
- Common Language Runtime (CLR)
- Windows APIs, DirectX

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
XNA Features

• 2D & 3D graphics support
  – Access to HLSL (High level shader language)
    • Pixel and vertex shaders
• Audio support
  – XACT cross-platform audio tool
• Controller and keyboard input
  – Xbox 360 controller
• Font support
• Content Pipeline
• Game save storage
• Networking
• ... and much more
Installing XNA Game Studio Express

• Follow instructions on pages linked from:
  – Also found on Tools page of course website

• Install Visual Studio
  – Visual Studio is an integrated development environment (editor/debugger/compiler)
  – Unless you currently use Visual Studio, you want “Visual C# 2008 Express”
    • XNA GSE will work with Visual Studio 2008 Professional if you have that installed instead

• Install XNA Game Studio 3.1
  – You want version 3.1, the latest version
  – The textbook covers 3.0
  – Version 3.1 is broadly similar to version 3.0
XNA Creator’s Club

- XNA Creator’s Club Website
  - http://creators.xna.com/
  - Community website for XNA GSE
  - Multiple complete games with source code
  - Many tutorials, and code examples
  - Very active discussion forums

- Creator’s Club Subscriptions
  - Can put game on Xbox 360
  - Access to premium content

- Trial membership - free
  - Available through Dream Spark or MSDNAA
    - Allows you to put game on Xbox 360

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
XNA Community Web Sites

• XNA Team Blog
  – blogs.msdn.com/xna/
  – Announcements from the XNA dev. Team

• Ziggyware
  – www.ziggyware.com
  – Developer-oriented XNA news
  – Recent contest for XNA tutorial articles
    • Winner: Skeel Keng-Siang Lee’s Introduction To Soft Body Physics

• XNA Development
  – www.xnadevelopment.com
  – XNA tutorials. See also the Links page for links to other quality XNA websites
Controllers

• XNA Game Studio Express allows you to use Xbox 360 controllers
  – Normal Xbox 360 controller is Bluetooth wireless, and is not recognized by the Windows Bluetooth manager
  – Hence, when developing game under Windows, won’t be able to test control scheme (bad)
• To create a game using Xbox 360 controller, need to:
  – Buy a corded Windows Xbox 360 controller (~$35 + shipping)
    • Google for “xbox 360 controller windows” for multiple online vendors
  – OR, buy an Xbox 360 wireless gaming receiver (~$20 + shipping)
    • allows wireless controller to work with Windows
  – Should buy now, so you have it ready for when you start programming
• Can also create a game that uses keyboard input
  – Would need to change control scheme to port to Xbox 360
Demonstration of Visual C# Express & XNA

- Demonstration of loading, compiling, and running one of the sample games for XNA Game Studio Express

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Homework

• Visit Creators Club website
• Download and install
  – Visual Studio C# 2008 Express
  – XNA Game Studio Express
• Compile and run a sample game
• Play around with starter kits

• Read Chapter 1 (Getting Started) in XNA 3.0
• Read in Programming C# 3.0
  – Chapter 1 (C# 3.0 and .NET 3.5)
  – Chapter 2 (Getting Started: "Hello World")
  – Chapter 3 (C# Language Fundamentals)
• Try one of the example code samples from the book for yourself in Visual C# 2008 Express
  – Get familiar with Visual C# 2008 environment
• Book is available online, via O’Reilly Safari
  – http://proquest.safaribooksonline.com/9780596527433
  – Use on-campus computer to access
 Intro to C#

- Slides adapted from Jim Whitehead’s “Intro to C#” at UCSC
Goals of the C# language

• A simple, modern, general-purpose object-oriented language
• Software robustness and programmer productivity
  – Strong type checking, array bounds checking, detection of use of uninitialized variables, source code portability, automatic garbage collection
• Useable in creating software components
• Ease of learning by programmers familiar with C++ and Java
• Usable for embedded and large system programming
• Strong performance, but not intended to compete with C or assembly language

Type II safety cans for flammables

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Brief history of C#

• Originated by Microsoft as a response to Java
  – Initial public release in 2000
• Language name inspired by musical note C#
  – A “step above” C/C++ (and Java)
  – Linux wags: D♭ (D-flat, same note, different name)
• Lead designers: Anders Hejlsberg, Scott Wiltamuth
  – Hejlsberg experience: Turbo Pascal, Borland Delphi, J++
• C# standardized via ECMA and ISO
  – However, Microsoft retains architectural control
Key language features

- Unified object system
  - Everything type is an object, even primitives
- Single inheritance
- Interfaces
  - Specify methods & interfaces, but no implementation
- Structs
  - A restricted, lightweight (efficient) type
- Delegates
  - Expressive typesafe function pointer
  - Useful for strategy and observer design patterns
- Preprocessor directives
Hello World example

class Hello
{
    static void Main()
    {
        // Use the system console object
        System.Console.WriteLine(“Hello, World!”);
    }
}

Creates a new object type (class) called Hello.

It contains a single method, called Main.

Main contains one line, which writes
“Hello, World!” on the display.

The method that performs this action is called WriteLine.

The WriteLine method belongs to the System.Console object.

The keyword “static” means that the method Main can be called even if there is no current instance
of the class. It’s a class method, not an instance method.

The line beginning with // is a comment, and does not execute.

Demonstration of creating Hello World inside Visual C# Express
Syntax

- Case-sensitive
- Whitespace has no meaning
  - Sequences of space, tab, linefeed, carriage return
- Semicolons are used to terminate statements (;)
- Curly braces {} enclose code blocks
- Comments:
  - /* comment */
  - // comment
  - /// <comment_in_xml>
    • Automatic XML commenting facility
Classes and Objects

• A class combines together
  – Data
    • Class variables
  – Behavior
    • Methods

• A key feature of object-oriented languages
  – Procedural languages, such as C, did not require clustering of data and behavior

• Class-instance distinction
  – Class defines variables & methods
  – Need to create instance of the class, called objects, to use variables & methods
  – Exception: static methods and variables
  – Analogy: a jelly bean mold (class) can be used to create a large number of jelly beans (objects, instances of the class)

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Defining a class

[attributes] [access-modifiers] class identifier [:base-class] {
  class-body
}

Simple example:

class A {
  int num = 0;    // a simple variable

  A (int initial_num) { num = initial_num; } // set initial value of num
}

• Attributes: used to add metadata to a class
  – Can safely be ignored
• Access modifiers: one of
  – public, private, protected, internal, protected internal
• Base-class
  – Indicates (optional) parent for inheritance
• Interfaces
  – Indicates (optional) interfaces that supply method signatures that need to be implemented in the class
• Class-body
  – Code for the variables and methods of the class
Inheritance

• Operationally
  – If class B inherits from base class A, it gains all of the
    variables and methods of A
  – Class B can optionally add more variables and methods
  – Class B can optionally change the methods of A

• Uses
  – Reuse of class by specializing it for a specific context
  – Extending a general class for more specific uses

• Interfaces
  – Allow reuse of method definitions of
    interface
  – Subclass must implement method
definitions
Inheritance Example

class A
{
    public void display_one()
    {
        System.Console.WriteLine("I come from A");
    }
}

class B : A
{
    public void display_two()
    {
        System.Console.WriteLine("I come from B, child of A");
    }
}

class App
{
    static void Main()
    {
        A a = new A();       // Create instance of A
        B b = new B();       // Create instance of B
        a.display_one();    // I come from A
        b.display_one();    // I come from A
        b.display_two();    // I come from B, child of A
    }
}
Visibility

• A class is a container for data and behavior
• Often want to control over which code:
  – Can read & write data
  – Can call methods
• Access modifiers:
  – Public
    • No restrictions. Members visible to any method of any class
  – Private
    • Members in class A marked private only accessible to methods of class A
    • Default visibility of class variables (but is good to state this explicitly)
  – Protected
    • Members in class A marked protected accessible to methods of class A and subclasses of A.

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Visibility Example

```java
class A {
    public int num_slugs;
    protected int num_trees;
    ...
}
class B : A {
    private int num_tree_sitters;
    ...
}
class C {
    ...
}
```

- **Class A can see:**
  - `num_slugs`: is public
  - `num_trees`: is protected, but is defined in A

- **Class B can see:**
  - `num_slugs`: is public in A
  - `num_trees`: is protected in parent A
  - `num_tree_sitters`: is private, but is defined in B

- **Class C can see:**
  - `num_slugs`: is public in A
  - Can’t see:
    - `num_trees`: protected in A
    - `num_tree_sitters`: private in B

---

Raindog, Flickr
www.flickr.com/photos/raindog/436176848/
Constructors

- **Use “new” to create a new object instance**
  - This causes the “constructor” to be called

- **A constructor is a method called when an object is created**
  - C# provides a default constructor for every class
    - Creates object but takes no other action
  - Typically classes have explicitly provided constructor

- **Constructor**
  - Has same name as the class
  - Can take arguments
  - Usually public, though not always
    - Singleton design pattern makes constructor private to ensure only one object instance is created

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Type System

- **Value types**
  - Directly contain data
  - Cannot be null
  - Allocated on the stack

- **Reference types**
  - Contain references to objects
  - May be null
  - Allocated on the heap

```
int i = 123;
string s = "Hello world";
```

Numeral type, by threedots
www.flickr.com/photos/threedots/115805043/

Slide adapted from “Introduction to C#”, Anders Hejlsberg
www.ecma-international.org/activities/Languages/Introduction%20to%20C-sharp.ppt

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Predefined Types

• C# predefined types
  – Reference object, string
  – Signed sbyte, short, int, long
  – Unsigned byte, ushort, uint, ulong
  – Character char (2 byte, Unicode)
  – Floating-point float, double, decimal
  – Logical bool

• Predefined types are simply aliases for system-provided types
  – For example, int == System.Int32

Slide from “Introduction to C#”, Anders Hejlsberg
www.ecma-international.org/activities/Languages/Introduction/FreeToUse/Csharp.ppt
Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Unusual types in C#

• **Bool**
  - Holds a boolean value, “true” or “false”
  - Integer values do not equal to boolean values
    • 0 does not equal false
    • There is no built-in conversion from integer to boolean

• **Decimal**
  - A fixed precision number up to 28 digits plus decimal point
  - Useful for money calculations
  - 300.5m
    • Suffix “m” or “M” indicates decimal

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Unified type system

• All types ultimately inherit from object
  – Classes, enums, arrays, delegates, structs, ...

• An implicit conversion exists from *any* type to type object
Unified Type System (Boxing)

- **Boxing**
  - Process of converting a value type to the type `object`
  - Wraps value inside a `System.Object` and stores it on the managed heap
    - Can think of this as allocating a “box”, then copying the value into it

- **Unboxing**
  - Extracts the value type from the object
  - Checks type of box, copies value out

```csharp
int i = 123;
object o = (object)i;
int j = (int)o;
```

Slide adapted from “Introduction to C#”, Anders Hejlsberg
www.ecma-international.org/activities/Languages/IntroductionToCSharp.ppt

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Variables

type variable-name [= initialization-expression];

Examples:

int number_of_slugs = 0;
string name;
float myfloat = 0.5f;
bool hotOrNot = true;

Also constants:

const int freezingPoint = 32;

• Variables must be initialized or assigned to before first use
• Class members take a visibility operator beforehand (private by default)
• Constants cannot be changed
Enumerations

enum identifier [: base-type]
{ enumerator-list}

Example:

denum Grades
{
   gradeA = 94,
   gradeAminus = 90,
   gradeBplus = 87,
   gradeB' = 84
}

• Base type can be any integral type (ushort, long) except for char
• Defaults to int
• Must cast to int to display in Writeln
  – Example: (int)g.gradeA

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Conditionals

if (expression) statement1
[else statement2]

Example:
if (i < 5) {
    System.Console.WriteLine("i is smaller than 5");
} else {
    System.Console.WriteLine("i is greater than or equal to 5");
}

• C# supports C/C++/Java syntax for “if” statement
• Expression must evaluate to a bool value
  – No integer expressions here
• == means “equal to” for boolean comparison
  – if (i == 5)  // if i equals 5
  – if (i = 5)  // error, since i = 5 is not a boolean expression

Adapted from Jim Whitehead’s slides from past CMPS 20/80K courses at UCSC
Switch statement

```csharp
switch (expression) {
    case constant-expression:
        statement(s);
    jump-statement
    [default: statement(s);]
}
```

- Alternative to if
- Typically use break
- Can use goto to continue to another case

Example:
```csharp
const int raining = 1;
const int snowing = 0;
int weather = snowing;
switch (weather) {
    case snowing:
        System.Console.WriteLine("It is snowing!");
        goto case raining;
    case raining:
        System.Console.WriteLine("I am wet!");
        break;
    default:
        System.Console.WriteLine("Weather OK");
        break;
}
```