Foreach Statement

foreach ( type identifier in array-or-collection )
{
    ...
}

• Iterates through all elements in an array, or collection type
• Creates identifier just for the scope of the statements inside the foreach
  – Holds the current element within the array/collection
• Very convenient when it can be used

string[] aStringArray = { “Cherry”, “Apple”, “Banana”, “Peach” };

// Sort elements
Array.Sort( aStringArray );

foreach (string s in aStringArray)
    System.Console.Write (“{0} : “, s);

// Output: “Apple : Banana : Cherry : Peach : ”
• Arrays have problem that you must know how many elements you want in advance
  – This is not always known
• List class is collection with variable size
  – Dynamically increases in size if needed
  – When an array reaches its capacity, need to create new array, and copy all elements from old array to new array
• Ugh!
Creating a List

List<type> listname

Example:

List<string> stringList = new List<string>();  // Create list of string. Don’t forget ()

stringList.Add ( “Quick” );
stringList.Add ( “Brown” );
stringList.Add ( “Fox” );

foreach (string s in myStringList) // Lists work with foreach
    System.Console.Write("{0} ", s);

• Add elements with Add() method
• Clear() removes all elements from list
• Remove() removes first element from list
• Sort() sorts the list
• Count property: number of elements in list
C# provides queue, stack, and dictionary

Queue: first-in, first-out
- Enqueue(), Dequeue(), Peek()

Stack: last-in, first-out
- Push(), Pop(), Peek()

Dictionary
- Holds set of key, value pairs
- Permits lookup of a value given the key
- Example use: extensible character attribute system
  - Keys: strings, names of attribute
  - Value: int, value of specific attribute
• Chapter 4 (Classes and Objects)
Chapter 9 (Arrays, Indexers, and Collections)
from pp. 155-176
in Programming C#
• Scaffolding for a simple XNA GSE game is created when you select a new game project in Visual C# Express
  – File ... New Project ... Windows Game (3.0)
  – Or File ... New Project ... Xbox 360 Game (3.0)
• Can fill-in this scaffolding to create your own game
• Creates a class (*myGameClass*) that includes
  – Constructor
  – Initialization
    • Initialize(), LoadContent()
  – Update
    • Update game state every clock tick
  – Draw
    • Create display every clock tick
• *Demonstration of XNA GSE scaffolding in Visual C# 2008 Express*
XNA GSE Game Scaffolding

Microsoft.Xna.Framework.Game

- # Initialize()
- # Run()
- # Tick()

myGame

- graphics: GraphicsDeviceManager
- content: ContentManager

+ myGame()
  # Initialize()
  # LoadContent(loadAllContent: bool)
  # UnloadContent(unloadAllContent: bool)
  # Update(gameTime: GameTime)
  # Draw(gameTime: GameTime)

GraphicsDeviceManager = new GraphicsDeviceManager(this);
Content.RootDirectory = "Content";
base.Initialize();
Update(gameTime);
Draw(gameTime);
XNA GSE Game Initialization

• Create new myGame
  – Call to constructor, myGame()
  – myGame.run()
    1. Initializes game, then,
    2. Runs the main game loop & processes events

• Initialization phase of run(),
  – The following methods are called on myGame
  – Initialize()
    1. call Initialize() on parent class
    2. Initialize your game state
      1. Create player object, create enemy objects, create object to hold main game state, etc.
  – LoadContent()
    • Method used to load textures, create SpriteBatches
XNA GSE Main Game Loop

• Time elapsed between each clock tick:
  – Fixed:
    • $1/60^{th}$ of a second (16.6667 milliseconds per tick)
    • myGame.IsFixedTimeStep = true
    • The default value
  – Variable:
    • Adjusts based on the time required to perform previous tick
    • myGame.IsFixedTimeStep = false

• Each clock tick
  – Run() calls Tick()
  – Tick() calls Update() then Draw()
    • You supply Update() and Draw()
Update() and Draw()

- **Update()**
  - Update the state of all objects
  - Receive input, move player avatar
  - Compute opponent AI, move opponent objects
  - Collision detection & consequences
  - Detect end-of-game or end-of-level condition

- **Draw()**
  - Re-create the on-screen scene using the up-to-date positions of player, opponent

- **Advice**
  - Avoid stuffing your entire game into the definition of these two methods
    - Methods become too big!
  - Have these methods call out to your player object, opponent objects, etc.
    - foreach (Opponent o in opponentList) o.update();
Getting a 2D Image to Appear on Screen

LoadContent()
1. Create a Texture
   – A bitmap image
2. Create a SpriteBatch
   – Collects all textures being drawn to screen

Draw()
3. Begin the SpriteBatch
4. Draw texture
   – Draw() is defined on a SpriteBatch
   – Adds texture to the SpriteBatch
5. End the SpriteBatch
   – Causes textures in SpriteBatch to be drawn to screen
Creating a Texture

• Create an instance of ContentManager
  – XNA GSE scaffolding does this for you
  – Content = new ContentManager(Services) in constructor

• Call Load<T> on ContentManager
  – For 2D sprites, type T is “Texture2D”
  – This loads an art asset that has been created by the Content Pipeline
    • In our case, conversion of a 2D bitmap image in PNG or JPG into XNA internal bitmap format
  – Give the pathname of the bitmap image (e.g., in PNG or JPG) to load
    • Path is relative to the “Content” directory of the Visual C# project
    • Note: normally need to escape slash in a string “\” → \n    • Can put “@” at beginning of string to make string “verbatim”
      – No need to escape slashes in this case
      – “\images\” is the same as @“\images\”
Example of creating a texture

• Create new bitmap image
  – In GIMP, Photoshop, etc.
  – Save to disk, then copy over to Visual C# project
    • Copy to Visual Studio 2010\Projects\{your project}\{your project}\Content
    • Go to Solution Explorer in Visual C# Express
    • Right click on Bolded Project Name
    • Add → Add Existing Item
    • Pick filename of new bitmap image file
    • Will now appear in the project file list
    • Verify that Content Pipeline processed file by building solution (F6)
      – Build > Build Solution

• Create a Texture2D, then load the bitmap image via the content manager:

  Protected Texture2D m_bullet = null;
  m_bullet = Content.Load<Texture2D>("mushi-bullet");
Once a texture has been made, how does this get displayed?

- Create a SpriteBatch
- Within a clock tick, begin the batch
  - Prepares the graphics device for drawing sprites
- Draw() the texture as part of the batch
- End the batch -- not an explicit call
  - Causes textures to be drawn to the screen
  - Restores device to how it was before the batch
- Typically this is performed in your game’s Draw() method
• **Draw() inside SpriteBatch is heavily overloaded**
  - 7 different choices

```csharp
protected override void LoadGraphicsContent(bool loadAllContent)
{
    if (loadAllContent)
    {
        m_batch = new SpriteBatch(graphics.GraphicsDevice);  // Initialize the sprite batch
        m_bullet = content.Load<Texture2D>("mushi-bullet");  // Create Texture2D
    }
}

protected override void Draw(GameTime gameTime)
{
    Vector2 loc = new Vector2(120, 120);    // Create Vector2 to give location of Texture2D
    m_batch.Begin();  // Start the batch
    m_batch.Draw(m_bullet, loc, Color.White);  // Add Texture2D to batch. Not yet on screen.
    m_batch.End();  // Now Texture2D is drawn to screen.
}
```
Tinting Sprites

• On previous slide, used Color.White in the Draw() method
  – This gives the tint of the sprite
  – White indicates use of the original colors
  – Can choose any other color to tint the sprite
    • Visual C# Express gives list of predefined colors
    • Can also defined a Vector3 to give RGB values

```csharp
protected override void Draw(GameTime gameTime)
{
    Vector2 loc = new Vector2(120, 120); // Create Vector2 to give location of Texture2D
    m_batch.Begin(); // Start the batch
    m_batch.Draw(m_bullet, loc, Color.Red); // Add Texture2D to batch. Has red tint.
    m_batch.End(); // Now Texture2D is drawn to screen.
}
```
Transparent Sprites

• It is possible to make a sprite partially opaque
  – Colors have RGB, and Alpha (opacity)
  – Use Vector4 to represent this
  – Create color by passing Vector4 into constructor

protected override void Draw(GameTime gameTime)
{
    Vector2 loc = new Vector2(120, 120);  // Create Vector2 to give location of Texture2D
    Vector4 v4Color = new Vector4(1.0f, 1.0f, 1.0f, 0.5f);  // Create Vector4 to create color w/opacity
    Color color = new Color(v4Color);  // Create color from v4Color
    m_batch.Begin();  // Start the batch
    m_batch.Draw(m_bullet, loc, color);  // Add Texture2D to batch. Is partially opaque
    m_batch.End();  // Now Texture2D is drawn to screen.
Other Sprite features

• Depth ordering
  – Draw some sprites in front of (behind) others to give depth of field effect

• Rotation
  – Can rotate sprite image to a specific angle

• Scaling
  – Can make sprites larger or smaller

• Animated sprites
  – Need to write code that cycles the animation yourself
  – Variant of batch.Draw() where you specify a specific rectangle within a Texture2D to draw

• Warp effects
  – Deform the Texture2D before placing on screen
Input
Upcoming Project Deliverable

- **Game Concept Document**
  - A compelling document that sells your game concept
  - Title page
    - Title of game, name of group, name of team members, sample artwork
  - Overview page
    - Table at top: game genre, platform (PC/XBox), team size
    - Key points section
      - Bulleted list of important elements of gameplay
      - Goal of game, what makes game unique, main characters, main fictional elements
      - Sample artwork image to give feel of the game
  - How your game addresses the Imagine Cup theme
  - Biographies
    - True, pocket biographies of each team member (1-2 paragraphs each) stressing experience that makes you a strong game designer/programmer/artist
  - 1-3 pages giving a textual description of the game
    - Fictional background, brief description of characters, goal of player in game, how does player interact with the game, brief description of levels, game audience, other important elements as needed.
  - 1-2 pages of sample conceptual artwork
    - Hand-drawn sketches are fine
- See template and evaluation criteria on course website
• XNA Framework supports three input sources
  – Xbox 360 controller
    • Wired controller under Windows
    • Wireless or wired for Xbox 360
    • Up to 4 at a time
  – Keyboard
    • Good default for Windows games
    • Xbox 360 also supports USB keyboards
  – Mouse
    • Windows only (no Xbox 360 support)

• Poll for input
  – Every clock tick, check the state of your input devices
  – Generally works OK for 1/60th second ticks
Digital vs Analog Controls

• Input devices have two types of controls

• Digital
  – Reports only two states: **on** or **off**
  – Keyboard: keys
  – Controller A, B, X, Y, Back, Start, D-Pad

• Analog
  – Report a **range of values**
  – XBox 360 controller: -1.0f to 1.0f
  – Mouse: mouse cursor values (in pixels)
<table>
<thead>
<tr>
<th>Input Device</th>
<th>Digital Buttons</th>
<th>Analog Controls</th>
<th>Vibration</th>
<th>Win?</th>
<th>Xbox?</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xbox 360 Controller</td>
<td>14</td>
<td>4</td>
<td>Yes</td>
<td>Yes (wired or wireless with adapter)</td>
<td>Yes (wireless or wired)</td>
<td>4</td>
</tr>
<tr>
<td>Keyboard</td>
<td>&gt;100</td>
<td>0</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Mouse</td>
<td>5</td>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>1</td>
</tr>
</tbody>
</table>
Xbox 360 Controller Input

• Every clock tick
  – Get state of controller
    • Call GetState() on GamePad class
    • Pass in PlayerIndex
      – PlayerIndex.One, PlayerIndex.Two, ...
      – Corresponds to lit region in “ring of light”
    • Returns a GamePadState structure
  – Check if controller is connected
    • IsConnected boolean in GamePadState
  – Check GamePadState for current state of digital and analog inputs
  – Recall that update() is called every clock tick
    • Get input in update(), or a method called from it
A **static** class
- Do not need to create an instance to use
- All methods are static

**GetState**
- Retrieve current state of all inputs on one controller

**SetVibration**
- Use to make controller vibrate

**GetCapabilities**
- Determine which input types are supported.
- Can check for voice support, and whether controller is connected.

```csharp
class GamePad
{
    public static GamePadCapabilities GetCapabilities(PlayerIndex playerIndex);
    public static GamePadState GetState(PlayerIndex playerIndex);
    public static GamePadState GetState(PlayerIndex playerIndex, GamePadDeadZone deadZoneMode);
    public static bool SetVibration(PlayerIndex playerIndex, float leftMotor, float rightMotor);
}
```
C# Structs

• A struct in C# is a lightweight alternative to a class
• Similar to class
  – Can have constructors, properties, methods, fields, operators, nested types, indexers
• Different from class
  – Struct does not support inheritance, or destructors
  – Is a value type (classes are reference types)
• Rule of thumb:
  – Use structs for types that are small, simple, similar in behavior to built-in types
public struct GamePadState
{
    public static bool operator !=(GamePadState left, GamePadState right);
    public static bool operator ==(GamePadState left, GamePadState right);
    public GamePadButtons Buttons { get; }
    public GamePadDPad DPad { get; }
    public bool IsConnected { get; }
    public int PacketNumber { get; }
    public GamePadThumbSticks ThumbSticks { get; }
    public GamePadTriggers Triggers { get; }
}

- Properties for reading state of the GamePad
  - Digital Input: Buttons, DPad
  - Analog Input: ThumbSticks, Triggers
  - Check connection state: IsConnected
  - PacketNumber
    - Number increases when gamepad state changes
    - Use to check if player has changed gamepad state since last tick
GamePadButtons Struct (Buttons)

GamePadState m_pad;                                  // create GamePadState struct
m_pad = GamePad.GetState(PlayerIndex.One);            // retrieve current controller state
if (m_pad.Buttons.A == ButtonState.Pressed)           // do something if A button pressed
if (m_pad.Buttons.LeftStick == ButtonState.Pressed)   // do something if left stick button pressed
if (m_pad.Buttons.Start == ButtonState.Pressed)       // do something if start button pressed

• Properties for retrieving current button state
  – A, B, X, Y
  – Start, Back
  – LeftStick, RightStick
    • When you press straight down on each joystick, is a button press
  – LeftShoulder, RightShoulder

• Possible values are given by ButtonState enumeration
  – Released – button is up
  – Pressed – button is down
GameDPad Struct (DPad)

GamePadState m_pad;                                         // create GamePadState struct
m_pad = GamePad.GetState(PlayerIndex.One);                   // retrieve current controller state
if (m_pad.DPad.Up == ButtonState.Pressed)                    // do something if DPad up button pressed
    // do something if DPad left button pressed

• Properties for retrieving current DPad button state
  – Up, Down, Left, Right

• Possible values are given by ButtonState enumeration
  – Released – button is up
  – Pressed – button is down
GamePadThumbsticks Struct (Thumbsticks)

GamePadState m_pad; // create GamePadState struct
m_pad = GamePad.GetState(PlayerIndex.One); // retrieve current controller state
if (m_pad.Thumbsticks.Left.X > 0.0f) // do something if Left joystick pressed to right
if (m_pad.Thumbsticks.Right.Y < 0.0f) // do something if Right joystick pressed down

• Each thumbstick has X, Y position
  – Ranges from -1.0f to 1.0f
    • Left (-1.0f), Right (1.0f), Up (1.0f), Down (-1.0f)
    • 0.0f indicates not being pressed at all
  – Represented as a Vector2
• So, have
• Joysticks typically have tiny deflection to left/right or up/down
  – Leads to “drift” if uncompensated

• **Dead-zone**
  – Region around 0.0f that is interpreted as not-moving
  – Allows joysticks to have a small amount of deflection without leading to drift

• Three ways to handle this
  – From GamePadDeadZone enumeration
  – **IndependentAxes**: X & Y each have separate dead zone (default)
  – **Circular**: X & Y combined before processing dead zone
  – **None**: No processing, application must determine
GamePadTriggers Struct (Triggers)

Each trigger ranges from 0.0f to 1.0f
- Not pressed: 0.0f
- Fully pressed: 1.0f
- Represented as a float

Have left and right triggers
- Properties: Left, Right

Demonstration of XNA Input Reporter utility
- creators.xna.com/en-US/utilities/inputreporter

GamePadState m_pad; // create GamePadState struct
m_pad = GamePad.GetState(PlayerIndex.One); // retrieve current controller state
if (m_pad.Triggers.Left != 0.0f) // do something if Left trigger pressed down
if (m_pad.Triggers.Right >= 0.95f) // do something if Right trigger pressed all the way down
Controller Vibration

- Can set the vibration level of the gamepad
  - Call `SetVibration()` on `GamePad` class
  - Pass controller number, left vibration, right vibration
    - Left motor is low frequency
    - Right motor is high-frequency

- Turn vibration full on, both motors
  - `GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);`

- Turn vibration off, both motors
  - `GamePad.SetVibration(PlayerIndex.One, 0f, 0f);`
• Every clock tick, poll state of keyboard
  – Call GetState() on Keyboard class
    • KeyboardState keyState = Keyboard.GetState()
    • Keyboard is a static class
  – Check if a specific key is pressed
    • if (keyState.IsKeyDown(Keys.keyname)) ...
    • Also, IsKeyUp(Keys.keyname)
    • Keys is an enumeration of keynames
    • Provides low-level access to keypress information
      – Can determine if right/left Shift key pressed, for example
  – Also, GetPressedKeys()
    • Returns array of keys currently pressed
    • If length of array is zero, no keys currently pressed
Mouse Input

- Every clock tick, poll state of the mouse
  - Call GetState on Mouse class
  - MouseState mouseState = Mouse.GetState();
  - Mouse is a static class
- MouseState contains a series of properties
  - X, Y: position of mouse (int)
  - LeftButton, MiddleButton, RightButton, XButton1, XButton2
    - Either Released or Pressed (ButtonState enumeration)
  - ScrollWheelValue (int)
    - Scroll wheel represents cumulative change over lifetime of the game
Wrapper Class

• What if you want to use the controller, if present, and the keyboard if not?
• Create an input wrapper class
  – Checks both controller and keyboard input
  – Has a series of properties to set/get current direction state
• Example:
  – If controller connected AND controller DPad up arrow pressed
    • Set wrapper’s up property to true
  – Else if keyboard Up key pressed
    • Set wrapper’s up property to true
  – Else
    • Set wrapper’s up property to false
• Rest of application checks input wrapper class up property
Reading

• Read Chapter 3 (User Input and Collision Detection) and 4 in *Learning XNA 4.0*

• Download and try XNA Input Recorder demo, if you have an Xbox 360 controller

• Try creating a simple XNA game to collect mouse/analog stick movement (Whackadot)