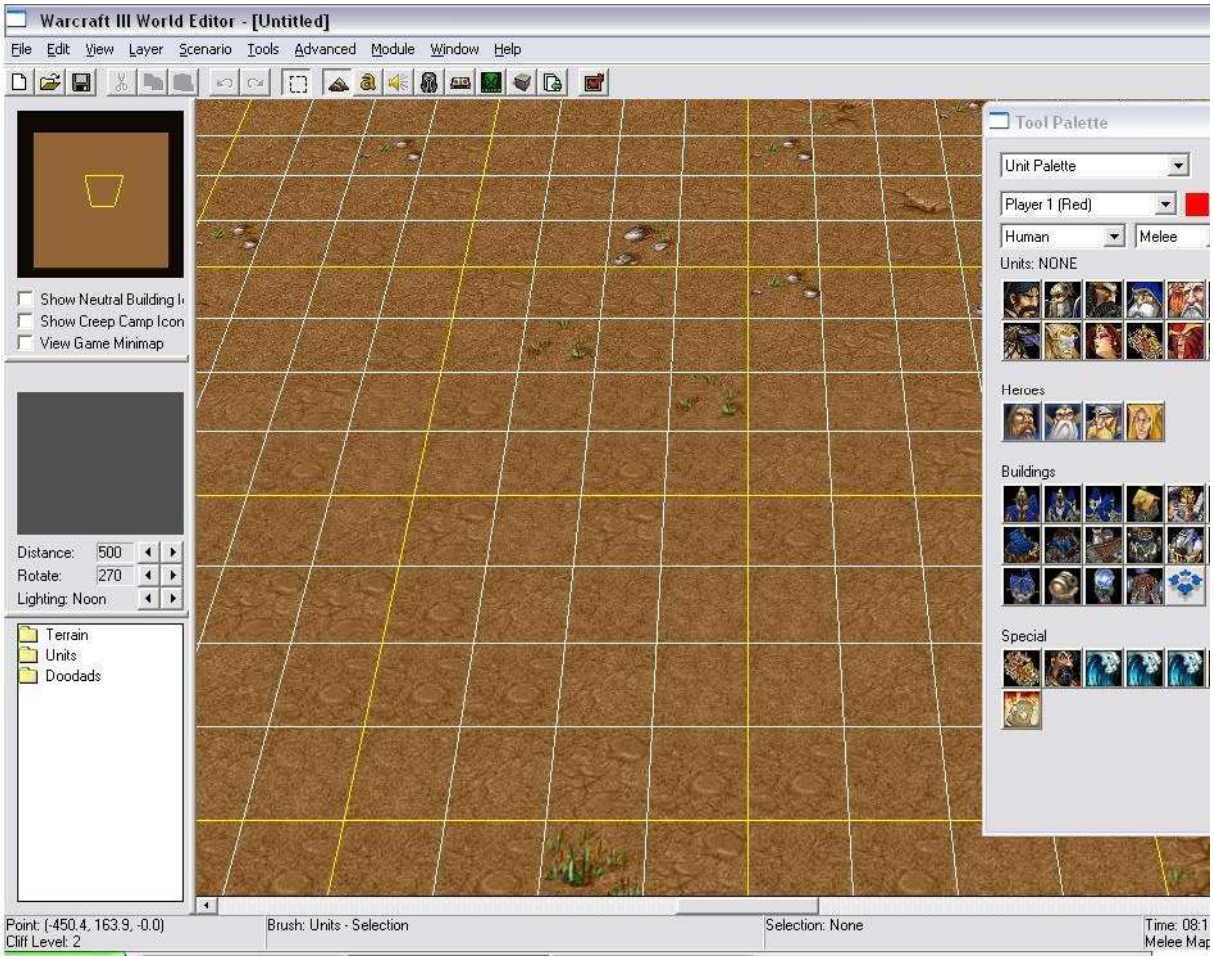
418383 การเขียนโปรแกรมเกม
User Interface Programming

ประมุข ขันเงิน

pramook@gmail.com

Game User Interface



Game User Interface

Game User Interface

- เราต้องการ control ลักษณะคล้ายกับของโปรแกรมประยุกต์ทั่วไป
  - ปุ่มกด
  - radio button
  - check box
  - drop down list box
  - sliders
  - ฯลฯ

วิธีการสร้าง User Interface

- เขียนเองทุกอย่าง
  - ☹
- ใช้ไลบรารีที่คนอื่นสร้างไว้
  - ใช้ WinForms
    - Control อยู่ข้างนอก XNA
    - เหมาะสำหรับใช้เวลาสร้าง editor ต่างๆ
  - ใช้ไลบรารีสำหรับสร้าง User Interface ในเกมโดยเฉพาะ
    - Control อยู่ใน XNA
    - เหมาะสำหรับใช้สร้าง user interface ในเกม

การใช้ XNA กับ WINFORMS

การทำงานของ XNA ปกติ

- เรา subclass Game
- Game สร้างหน้าต่างหนึ่งหน้าต่างสำหรับเกมของคุณ
- เรา override เมธอด Update และ Draw

- บางกรณีเราอาจพบว่า framework นี้มีข้อจำกัด
  - บางครั้งคุณอยากจะสร้างหน้าต่างเอง
  - บางครั้งคุณอาจจะอยากสร้าง editor สำหรับเกม

Microsoft.Xna.Framework.Game

- XNA ประกอบด้วย assembly หลายๆ ตัว
  - Microsoft.Xna.Framework
  - Microsoft.Xna.Framework.Graphics
  - Microsoft.Xna.Framework.Content
  - ฯลฯ
- Microsoft.Xna.Framework.Game มีคลาส Game ที่เป็นโค้ดระดับสูงที่ใช้ assembly อื่นในการสร้าง application สำหรับเกมโดยเฉพาะ

การรันโค้ดเกมด้วยวิธีอื่น

- คุณต้องสร้างคลาสที่ทำหน้าที่แทน Game
- ในที่นี้เราจะสร้าง control ที่โค้ด XNA สามารถวาดรูปใส่ได้
- แล้วเอา control นี้ไปเป็นส่วนประกอบของโปรแกรมที่ใช้ Windows Forms ธรรมดา

- โค้ดตัวอย่างที่อยู่ใน WinFormsGraphicsSample_4_0 จะมีคลาส GraphicsDeviceControl สำหรับทำหน้าที่นี้โดยเฉพาะ

GraphicsDeviceControl

- สืบเชื้อสายมาจาก System.Windows.Form.Cotnrol
  - มีความสามารถในการวาดรูปตัวเองบนหน้าจอ
  - ใช้ GraphicsDevice ของ XNA ในการวาดรูป
- สามารถใช้มันใน WinForms Designer ได้
  - แต่เวลาใช้จะวาดรูปตัวเองเป็นพื้นที่ว่างสีฟ้าเฉยๆ

สร้างโปรแกรม Windows Forms ที่ใช้ XNA ได้

- สร้าง Windows Forms Application แทนการสร้าง XNA Game
- เพิ่ม Reference ต่อไปนี้
  - Microsoft.Xna.Framework
  - Microsoft.Xna.Framework.Graphics
- คัดลอกไฟล์
  - GraphicsDeviceControl.cs
  - GraphicsDevice.cs
  - ServiceContainer.cs

สร้างโปรแกรม Windows Forms ที่ใช้ XNA ได้

- สร้าง subclass ของ GraphicsDeviceControl ใหม่ โดย override เมธอด
  - Initialize
  - Draw

การจัดการ GRAPHICS DEVICE

การจัดการ GraphicsDevice

- ในโปรแกรมหนึ่งๆ อาจมี GraphicsDeviceControl อยู่ได้หลายๆ ตัว
- แต่เพื่อให้โปรแกรมมีประสิทธิภาพ ควรจะมี GraphicsDevice เพียงแค่ตัวเดียวเท่านั้น

GraphicsDeviceService

- คลาสที่ทำการจัดการ GraphicsDevice
- สร้างมันขึ้นมาเพียงตัวเดียว แล้วแจกจ่ายให้ GraphicsDeviceControl หลายๆ ตัวใช้

- มีเมธอด AddRef สำหรับใช้เพิ่ม GraphicsDeviceControl ตัวใหม่
  - เมื่อ AddRef ถูกเรียกครั้งแรกจะสร้าง GraphicsDevice ขึ้นมาใหม่
  - แต่เมื่อถูกเรียกครั้งต่อไปจะไม่สร้างใหม่ให้

GraphicsDeviceService

- มีเมธอด Release สำหรับให้ GraphicsDeviceControl เรียกเมื่อมันจะเลิกใช้ GraphicsDevice
  - ถ้าเมื่อเรียกแล้วจำนวน GraphicsDeviceControl ที่ใช้เป็น 0 แล้ว GraphicsDeviceService จะทำลาย GraphicsDevice ทิ้ง

GraphicsDevice ขนาดแตกต่างกัน

- เมื่อมี GraphicsDeviceControl หลายๆ ตัว
  แต่ละตัวอาจมีพื้นที่แตกต่างกัน
- แต่ GraphicsDevice มีเพียงแค่ตัวเดียว
  และ GraphicsDeviceControl ทุกตัวต้องใช้มันร่วมกัน
- แล้วเจ้า GraphicsDevice ตัวนี้ควรจะมี back buffer ขนาดเท่าใดดี?

GraphicsDevice ขนาดแตกต่างกัน

- โค้ดตัวอย่างจะกำหนดขนาด back buffer ให้มีขนาดใหญ่กว่า GraphicsDeviceControl ที่ใหญ่ที่สุด
- เวลา GraphicsDeviceControl ต่างๆ จะใช้ GraphicsDevice มันจะต้องกำหนด “หน้าต่าง” ใน GraphicsDevice ที่มันจะใช้ก่อนเรียกใช้จริง
- หลังจากวาดรูปใส่ GraphicsDevice เสร็จแล้วก็ต้อง copy ข้อมูลเฉพาะส่วนที่สนใจนำไปแสดงผล

BeginDraw → Draw → EndDraw

- วาดรูปตัวเองโดยของ GraphicsDeviceControl จึงแบ่งเป็นสามขั้นตอน และแต่ละขั้นตอนมีเมธอดของตัวเอง
- GraphicsDeviceControl.BeginDraw
  - กำหนดขนาดของ GraphicsDevice ให้ใหญ่กว่าพื้นที่ตัวเอง
  - กำหนดบริเวณของ GraphicsDevice ที่จะใช้จริง (Viewport)
- GraphicsDeviceControl.Draw
  - วาดรูปตัวเองตามที่ผู้ใช้ต้องการ
- GraphicsDevice.EndDraw
  - เรียก GraphicsDevice.Present เพื่อเอารูปที่วาดใส่ GraphicsDevice ไปแสดงผลในบริเวณของตัวเอง

การอ่าน CONTENT ใน WINFORMS APPLICATION

อ่าน Content ใน WinForms App

- ถ้าคุณต้องการอ่าน content ที่สร้างด้วย content pipeline คุณต้องทำสองอย่างต่อไปนี้
  - Content ที่คุณจะใช้จะต้องถูก build ไปพร้อมกับ project ของคุณ
  - ต้องสร้าง ContentManager

Build Content พร้อมกับ App

- สร้าง content project ใหม่โดยเลือก Add New Item → Empty Content Project
- ใส่ content ที่คุณต้องการใช้ใน content project ที่สร้าง
- แต่ content project ไม่สามารถ build ตัวเองได้
- คุณต้องสร้าง XNA Game Library ขึ้นมาอันหนึ่ง
- แล้วเพิ่ม content project เข้าใน game library โดยเพิ่มมันเข้าใน Content Reference ของ game library
- หลังจากนั้นจึงเพิ่ม reference ของ game library ลงใน application

สร้าง ContentManager

- ContentManager ต้องการ GraphicsDevice สำหรับอ่านข้อมูลทางกราฟิกส์ต่างๆ (เช่น Texture2D)
- เวลา ContentManager ต้องการใช้ GraphicsDevice มันจะไปเรียกหา GraphicsDevice จากคลาสที่ implement IGraphicsDeviceService
- ใน GraphicsDeviceControl จะมี property Services ที่เราสามารถไปดึง IGraphicsDeviceService ออกมาได้
- ดูวิธีการสร้าง ContentManager ใน constructor ของ SpriteFontControl

อนิเมชันใน WINFORMS APPLICATION

อนิเมชันใน WinForms Application

- Application ที่เขียนด้วย Windows Forms ทั่วไปจะไม่มีภาพเคลื่อนไหว
  - Control ต่างๆ ส่วนใหญ่จะอยู่นิ่ง
  - จะถูกปลุกให้ตื่นขึ้นมาเพื่อวาดรูปตัวเองใหม่เมื่อผู้ใช้ป้อน input
- ต่างกับเกม
  - ต้องมีการวาดรูปใหม่อยู่อย่างต่อเนื่อง
  - เกมใน XNA ส่วนใหญ่จะวาดรูปแบบนี้
- วิธีการวาดรูปของ WinForms กับของ XNA จึงเข้ากันไม่ได้

เมื่อใช้ XNA กับ WinForms

- เมื่อคุณสร้าง GraphicsDeviceControl แล้วจะต้องเลือกว่าจะใช้วิธีการวาดรูปตัวเองแบบไหน
- ในโค้ดตัวอย่างมี control ที่ใช้วิธีการวาดทั้งสองแบบ
  - SpriteFontControl ใช้วิธีวาดแบบ WinForms
  - TriangleControl ใช้วิธีวาดแบบ XNA

วิธีวาดแบบ WinForms

- หากต้องการใช้วิธีวาดแบบ WinForms เวลา subclass GraphicsDeviceControl ก็ไม่ต้องเขียนโค้ดเพิ่มเติม
  - แค่ override Initialize กับ Draw ตามปกติ
  - Draw จะถูกเรียกเมื่อ control จะต้องวาดตัวเองโดยอัตโนมัติ

วิธีการวาดแบบ WinForms

- ถ้าหากต้องการให้ control ทำการวาดตัวเองใหม่ทุกๆ ช่วงเวลาที่กำหนด ให้
  - สร้าง Timer ไว้สำหรับจับเวลาที่ผ่านไป
  - ตั้งค่า Timer.Interval ให้ตรงกับช่วงเวลาที่ต้องการ
  - สร้างเมธอดที่จะถูกเรียกเมื่อ Timer ตีสัญญาณเวลา
  - แล้วเพิ่มเมธอดนั้นเข้าใน event Timer.Tick
  - หลังจากนั้นเรียก Timer.Start()
- โค้ดข้างบนทั้งหมดควรใส่ไว้ใน constructor ของ subclass ของ GraphicsDeviceControl ที่คุณเขียน

วิธีการวาดแบบ WinForms

- โค้ดตัวอย่าง

```
timer = new Timer();
timer.Interval =
   (int)TargetElapsedTime.TotalMilliseconds;
timer.Tick += Tick;
timer.Start();
```

วิธีการวาดแบบ WinForms

- ข้อดี
  - เข้ากับส่วนอื่นของระบบ UI ได้ดีที่สุด
  - สามารถเอาเวลาระหว่างที่ Timer ตีสัญญาณไปใช้ทำอย่างอื่นได้
  - ใช้งานได้ดีถ้ามี control หลายตัวที่ต้องทำอนิเมชันพร้อมกัน
- ข้อเสีย
  - อาจไม่ได้ภาพเคลื่อนไหวที่ลื่นไหลที่สุด
  - Timer อาจตีสัญญาณช้ากว่ากำหนดถ้าระบบกำลังทำงานหนัก
  - Timer มีความแม่นยำจำกัด

วิธีการวาดแบบ XNA (1)

- SpinningTriangleControl ทำการวาดรูปบ่อยที่สุดเท่าที่จะทำได้โดยการเพิ่ม handle ใส่ event Application.Idle
  - Application.Idle += Tick;
- Application.Idle จะถูกปล่อยออกมาด้วย Win32 message queue ทุกครั้งที่ message queue ว่าง
- ถ้าจะใช้เมธอดนี้ในการวาดรูป เราจะต้องเรียกเมธอด Invalidate ใน Tick

วิธีการวาดแบบ XNA (1)

- กลไกการทำงาน
  - Win32 message queue ยิง Application.Idle
  - Invalidate ถูกเรียก
  - Invalidate ส่ง WM_PAINT ไปยัง control
  - WM_PAINT ถูกประมวลผล ทำให้ control วาดตัวเองใหม่
  - ไม่มี message ค้างใน queue ทำให้ Application.Idle ถูกเรียกใหม่
- ถ้าใน event handler ไม่มีการเรียก Invalidate แล้วจะมี Application.Idle ส่งมาเพียงครั้งเดียวเท่านั้น

วิธีการวาดแบบ XNA (1)

- ข้อดี
  - เขียนโค้ดสั้น
- ข้อเสีย
  - อาจไม่ได้อนิเมชันที่ลื่นไหลที่สุด เนื่องจากมีช่วงเวลาระหว่างที่ Application.Idle ถูกส่งกับการวาดรูปจริง

วิธีการวาดแบบ XNA (2)

- เราสามารถทำให้โปรแกรมของเราเช็ค Win32 message queue อย่างต่อเนื่อง เพื่อให้ตอบสนองต่อ message ได้อย่างรวดเร็วยิ่งขึ้น
- ขั้นแรกให้ใส่ event handler ใหม่ให้ Application.Idle
  - Application.Idle += TickWhileIdle;
- TickWhileIdle จะเข้า infinite loop แล้วเรียก Tick ทันทีเมื่อ message queue ว่าง

วิธีการวาดแบบ XNA (2)

```
void TickWhileIdle(object sender, EventArgs e)
  {
     NativeMethods.Message message;

     while (!NativeMethods.PeekMessage(out message,
  IntPtr.Zero, 0, 0, 0))
     {
        Tick(sender, e);
     }
  }
```

วิธีการวาดแบบ XNA (2)

- NativeMethods.PeekMessage เป็นเมธอดที่เราเขียนขึ้นมาเพื่อดูว่ามี message ใน Win32 message queue หรือไม่

วิธีการวาดแบบ XNA (2)

```
static class NativeMethods
  {
     [StructLayout(LayoutKind.Sequential)]
     public struct Message
     {
        public IntPtr hWnd;
        public uint Msg;
        public IntPtr wParam;
        public IntPtr lParam;
        public uint Time;
        public System.Drawing.Point Point;
     }

     [DllImport("User32.dll")]
     [return: MarshalAs(UnmanagedType.Bool)]
     public static extern bool PeekMessage(out Message message,
         IntPtr hWnd, uint filterMin, uint filterMax, uint flags);
  }
```

วิธีการวาดแบบ XNA (2)

- ข้อดี
  - ได้อนิเมชันที่ลื่นไหลที่สุด
- ข้อเสีย
  - เข้ากับระบบส่วนอื่นไม่ได้เนื่องจากแย่งเวลาเขาไปใช้หมด
  - เอาเวลาการทำงานของ CPU ไปใช้ 100%
  - เกิดปัญหาถ้ามี control ที่ต้องทำอนิเมชันพร้อมๆ กัน

เมธอด Tick

- เพื่อให้อนิเมชันลื่นไหล เมธอด Tick จะต้องทำการ update สถานะของเกมและวาดรูปในอัตราที่คงที่
- เทคนิคการวาดทั้งสองแบบของเราที่ผ่านไม่ได้รับประกันอัตราการวาด
- ดังนั้นใน Tick จะต้องมีการเช็คนาฬิกาว่าเวลาผ่านไปเท่าไหร่แล้ว แล้วจึงเรียก update เพื่อเวลาผ่านไปมากกว่าคาบที่ต้องทำการ update

เมธอด Tick

```
Stopwatch stopWatch = Stopwatch.StartNew();

readonly TimeSpan TargetElapsedTime =
 TimeSpan.FromTicks(TimeSpan.TicksPerSecond / 60);
readonly TimeSpan MaxElapsedTime =
 TimeSpan.FromTicks(TimeSpan.TicksPerSecond / 10);

TimeSpan accumulatedTime;
TimeSpan lastTime;
```

เมธอด Tick

```
void Tick(object sender, EventArgs e)
{
    TimeSpan currentTime = stopWatch.Elapsed;
    TimeSpan elapsedTime = currentTime - lastTime;
    lastTime = currentTime;

    if (elapsedTime > MaxElapsedTime)
        elapsedTime = MaxElapsedTime;
    accumulatedTime += elapsedTime;

    bool updated = false;
    while (accumulatedTime >= TargetElapsedTime) {
        Update();
        accumulatedTime -= TargetElapsedTime;
        updated = true;
    }

    if (updated)
        Invalidate();
}
```

IN-GAME USER INTERFACE

ไลบรารีสำหรับสร้าง User Interface

- ขณะนี้มีไลบรารีสำหรับสร้าง user interface ใน XNA หลายตัว
  - xWinForms (http://sourceforge.net/projects/xwinforms/)
  - BlackStar GUI (http://blackstar.codeplex.com/)
  - XNA Simple GUI (http://simplegui.codeplex.com/)
  - Windows System for XNA (http://wsx.codeplex.com/)
  - Nuclex Framework (http://nuclexframework.codeplex.com/)
- Library อื่นๆ
  - หาข้อมูลเพิ่มได้จาก http://forums.create.msdn.com/forums/t/15274.aspx

Nuclex Framework

- เราจะใช้ Nuclex Framework
  - เขียนสำหรับ XNA 4.0
  - มีการ update เมื่อไม่นานมานี้
    - โปรเจคอื่นๆ หลายตัวรู้สึกจะเลิกทำกันไปหลังจากปี 2009 ก็เยอะ
- การใช้ Nuclex Framework
  - เว็บใช้แนะนำให้ดาวน์โหลด source code แล้วเพิ่มโปรเจคที่ต้องการเข้าในเกมของคุณ
  - ในกรณีการทำ user interface เราจะใช้
    - Nuclex.Input
    - Nuclex.UserInterface

GuiManager

- Nuclex.GuiManager เป็นคลาสที่ใช้ในการแสดงผล GUI control ทั้งหมด
  - คุณสามารถเพิ่ม control อื่นๆ ให้เป็น “ลูก” ของ GuiManager
  - เมื่อ GuiManager.Draw ถูกเรียก มันจะวาดลูกของมันทั้งหมด
- GuiManager เป็น DrawableGameComponent

DrawableGameComponent

- คลาส DrawableGameComponent เป็นกลไลของ XNA ในการแยกเกมออกเป็นส่วนๆ ที่เป็นอิสระจากกัน
  - สามารถเอาไปเพิ่มให้กับ Game เพื่อให้เกมมีความสามารถใหม่
  - สามารถใช้ DrawableGameComponent เดียวกันกับเกมหลายๆ เกมได้

DrawableGameComponent

- เวลาสร้าง DrawableGameComponent จะต้อง override เมธอดต่อไปนี้
  - Initialize
  - LoadContent
  - UnloadContent
  - Update
  - Draw
- คล้ายๆ กับสร้าง Game และ Screen ใหม่

DrawableGameComponent

- DrawableGameComponent กับ Screen ไม่เหมือนกัน
  - ณ เวลาหนึ่งจะมี Screen เพียงแค่ Screen เดียว
  - คุณ “เปลี่ยน” Screen
  - ในเวลาหนึ่งจะมี DrawableGameComponent มากกว่าหนึ่งตัว
  - คุณ “เพิ่ม” DrawableGameComponent เมื่อต้องการใช้
  - คุณ “ลบ” DrawableGameComponent เมื่อเลิกใช้

DrawableGameComponent

- สิ่งที่เรามักจะได้ DrawableGameComponent ทำ
  - เคอร์เซอร์ของเมาส์
  - User Interface
- ผมไม่ค่อยชอบใช้ DrawableGameComponent
  - ควบคุมการ Update และ Draw ของมันยาก เพราะ Game จะเรียกฟังก์ชันเหล่านี้โดยอัตโนมัติ

DrawableGameComponent

- ถ้าต้องการเพิ่ม DrawableGameComponent เข้าในเกม ให้สั่ง Components.Add(<component ที่ต้องการ>) ใน constructor ของเกม

```
public Game1()
{
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";

        Components.Add(new MyComponent(this));
}
```

- เมธอดต่างๆ ของ DrawableGameComponent จะถูกเรียกเมื่อเมธอดเดียวกันของ Game ถูกเรียก

การใช้ GuiManager

- สร้างฟิลด์ชนิด GuiManager ในคลาส Game
- ใน constructor ของ Game
  - สร้าง GuiManager
  - เพิ่ม GuiManager เข้าใน Components

การใช้ GuiManager

```
public class UserInterfaceDemoGame : Game {

  public UserInterfaceDemoGame() {
    this.graphics = new GraphicsDeviceManager(this);
    this.gui = new GuiManager(this);
    Components.Add(this.gui);
    IsMouseVisible = true;
  }

  // …

  private GuiManager gui;
}
```

Screen

- GuiManager ต้องมี Screen ของมัน
  - ชื่อซ้ำกันกับ Screen ของ Game แต่คนละตัว
- Screen มีหน้าที่จัดการสถานะของ GUI ต่างๆ
- เราอาจมี Screen หลายๆ อันถ้าเรามีระบบ GUI หลายตัว
- แต่ส่วนมากจะมีแค่ตัวเดียว

Desktop Control

- ระบบ GUI จะมีต้นไม้ของ control
  - Control ตัวหนึ่งสามารถมี control ตัวอื่นเป็นลูก
- Control ที่อยู่ระดับบนสุดเรียกว่า *Desktop Control*
  - Control นี้เป็นลูกของ screen
  - Desktop Control เป็นพื้นที่ที่ไม่มีสี
  - ถ้าคุณเอา control อื่นไปใส่เป็นลูกของ desktop control มันจะปรากฏอยู่บนหน้าจอเกมโดยตรง

การใช้ GuiManager

- ในฟังก์ชัน Initialize ของ Game
  - สร้าง Screen โดยให้ Viewport ของ GraphicsDevice เป็น argument
  - ในขั้นนี้คุณอาจจะกำหนดขอบเขตของ Desktop Control ไม่ให้ใหญ่เกินไปได้

การใช้ GuiManager

```
protected override void Initialize() {

  Viewport viewport = GraphicsDevice.Viewport;
  Screen mainScreen = new Screen(viewport.Width, viewport.Height);
  this.gui.Screen = mainScreen;

  mainScreen.Desktop.Bounds = new UniRectangle(
    new UniScalar(0.1f, 0.0f), new UniScalar(0.1f, 0.0f), // x and y
    new UniScalar(0.8f, 0.0f), new UniScalar(0.8f, 0.0f) // width and height
  );

  base.Initialize();
}
```

UniScalar

- การบอกตำแหน่งของ Nuclex Framework ใช้กลไกคล้ายกับของ CeGUI (http://www.cegui.co.uk)
- Coordinate ทุกตัวแทนด้วยคลาส UniScalar
- UniScalar มีส่วนประกอบสองส่วน
  - Fractional part = อัตราส่วนเท่าไหร่ของความกว้างของ “พ่อ”
  - Offset = ห่างจากตำแหน่งที่กำหนดด้วย fractional part มากี่พิกเซล
- ตัวอย่าง
  - new UniScalar(0.5f, 50) = ตำแหน่งที่ห่างจากจุดกึ่งกลางของพ่อมา 50 พิกเซล

UniScalar

- ดังนั้น

```
new UniRectangle(
    new UniScalar(0.1f, 0.0f), new UniScalar(0.1f, 0.0f), // x and y
    new UniScalar(0.8f, 0.0f), new UniScalar(0.8f, 0.0f) // width and
    height
);
```

จึงเป็นสี่เหลี่ยมที่มีขนาด 80% ของ control พ่อ

และอยู่ตรงกลาง control พ่อพอดี

สร้าง Control อื่นๆ

- สร้าง control อื่นๆ เพื่อมาใส่เป็นลูกของ desktop control
- ในกรณีนี้เราจะสร้าง WindowControl ซึ่งเป็นหน้าต่างซึ่งข้างในสามารถบรรจุ control อื่นๆ ได้

สร้าง Control อื่นๆ

```
public partial class DemoDialog : WindowControl {

  public DemoDialog() {
    InitializeComponent();
  }

}
```

สร้าง Control อื่นๆ

```
partial class DemoDialog {

  private Nuclex.UserInterface.Controls.LabelControl helloWorldLabel;
  private Nuclex.UserInterface.Controls.Desktop.ButtonControl okButton;
  private Nuclex.UserInterface.Controls.Desktop.ButtonControl cancelButton;

  private void InitializeComponent() {
    this.helloWorldLabel = new Nuclex.UserInterface.Controls.LabelControl();
    this.okButton = new Nuclex.UserInterface.Controls.Desktop.ButtonControl();
    this.cancelButton = new Nuclex.UserInterface.Controls.Desktop.ButtonControl();

    // helloWorldLabel
    this.helloWorldLabel.Text = "Hello World! This is a label.";
    this.helloWorldLabel.Bounds = new UniRectangle(10.0f, 15.0f, 110.0f, 30.0f);
```

สร้าง Control อื่นๆ

```
    // okButton
    this.okButton.Bounds = new UniRectangle(
      new UniScalar(1.0f, -180.0f), new UniScalar(1.0f, -40.0f), 80, 24
    );

    // cancelButton
    this.cancelButton.Bounds = new UniRectangle(
      new UniScalar(1.0f, -90.0f), new UniScalar(1.0f, -40.0f), 80, 24
    );

    // DemoDialog
    this.Bounds = new UniRectangle(100.0f, 100.0f, 512.0f, 384.0f);
    Children.Add(this.helloWorldLabel);
    Children.Add(this.okButton);
    Children.Add(this.cancelButton);
  }
}
```

Children.Add

- เมื่อสร้าง control เสร็จแล้วก็ให้ไปเพิ่มมันเป็นลูกของ control อื่นโดยการเรียกเมธอด Add ของ property Children
- เช่น เราสามารถเพิ่ม DemoDialog เป็นลูกของ Desktop Control ได้ดังต่อไปนี้

```
protected override void Initialize() {

  // ...code from previous step...

  // Next, we add our demonstration dialog to the screen
  mainScreen.Desktop.Children.Add(new DemoDialog());

  base.Initialize();
}
```

การจัดการกับ Event ของ Control ต่างๆ

- เช่นเดียวกับ WinForms control control ของ Nuclex Framework มี event ที่เราสามารถกำหนด ให้มันได้
- เช่น คลาส ButtonControl มี event Pressed ที่จะถูกส่งทุกครั้งเมื่อปุ่มถูกกด
- เราสามารถเพิ่ม handler ที่มี signature เป็น

  void Handler(object sender, EventArgs arguments)

  ให้เป็นตัวจัดการ event ได้

การจัดการกับ Event ของ Control ต่างๆ

```
private void createDesktopControls(Screen mainScreen) {

    // Button through which the user can quit the application
    ButtonControl quitButton = new ButtonControl();
    quitButton.Text = "Quit";
    quitButton.Bounds = new UniRectangle(
      new UniScalar(1.0f, -80.0f), new UniScalar(1.0f, -32.0f), 80, 32
    );
    quitButton.Pressed += delegate(object sender, EventArgs arguments) { Exit(); };
    mainScreen.Desktop.Children.Add(quitButton);

}
```