418383 การโปรแกรมเกม
การบรรยายครั้งที่ 3

ประมุข ขันเงิน

pramook@gmail.com

XNA

- ไลบรารีภาษา C# ของ Microsoft สำหรับเขียนเกม
  - Platform เป้าหมาย: PC, Xbox, Zune
  - ความสามารถ:
    - วาดภาพ 2 มิติและ 3 มิติ
    - เล่นเสียงและวิดีโอ
    - จัดการ content ของเกม
    - รับและประมวลผล input จาก keyboard, mouse, joystick

XNA

- ช่วยทำให้การเขียนเกมง่ายขึ้น
  - ไม่ต้องกังวลเรื่อง hardware
  - ไม่ต้องเขียนโค้ดเพื่อวาดภาพ อ่านข้อมุลเข้า หรือเล่นเสียง
- แต่ก็ยังต้องเขียนมากพอสมควร
- เนื่องจากไม่มีองค์ประกอบพื้นฐานสำหรับสร้างเกมที่มีความซับซ้อน
  - ไม่มี collision detection
  - ไม่มี physics engine
  - ไม่มีความสามารถในการจัดการข้อมูล 2 มิติหรือ 3 มิติระดับสูง

การติดตั้ง

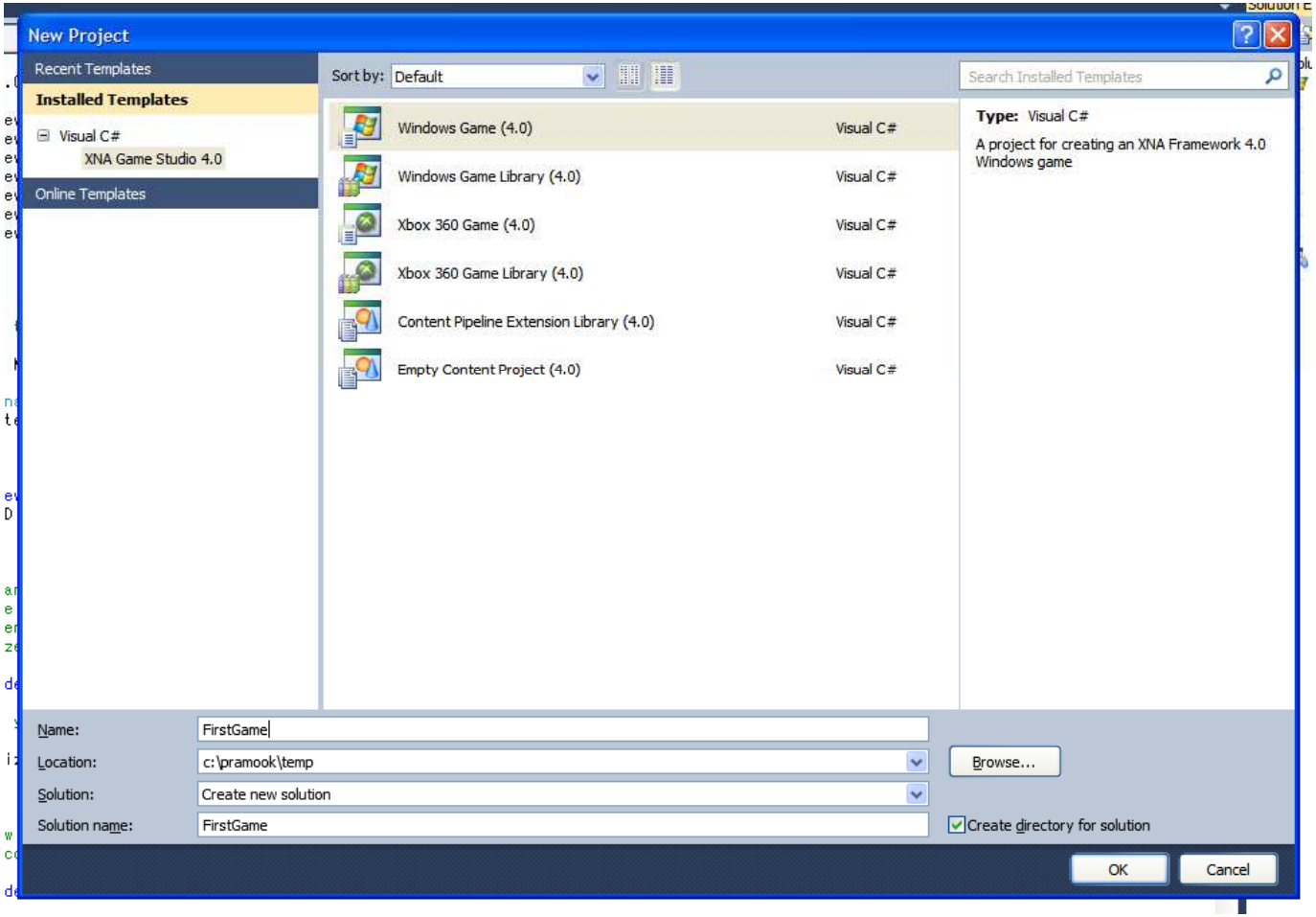
- ดาวน์โหลด Visual C# Express 2010 http://www.microsoft.com/express
- ดาวน์โหลด Microsoft XNA Game Studio 4.0

- ลง Visual C# ให้เสร็จก่อนลง XNA Game Studio

สร้างเกมใหม่

- เปิด Visual C# 2010
- เลือกเมนู File → New Project…
- ในช่องทางซ้าย เลือก XNA Game Studio 4.0
- ในช่องทางขวา เลือก Windows Games (4.0)
- ตั้งชื่อเกมใน Name
- ตั้งชื่อ Solution (กลุ่มของ project ที่เกี่ยวข้องกัน) ใน Solution Name

สร้างเกมใหม่


New Project
Recent Templates
Installed Templates
Visual C#
XNA Game Studio 4.0
Online Templates
Sort by: Default
Search Installed Templates
Windows Game (4.0) Visual C#
Windows Game Library (4.0) Visual C#
Xbox 360 Game (4.0) Visual C#
Xbox 360 Game Library (4.0) Visual C#
Content Pipeline Extension Library (4.0) Visual C#
Empty Content Project (4.0) Visual C#
Type: Visual C#
A project for creating an XNA Framework 4.0 Windows game
Name: FirstGame
Location: c:\pramook\temp
Solution: Create new solution
Solution name: FirstGame
Browse...
Create directory for solution
OK
Cancel

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame → เกมใหม่ที่เราเพิ่งจะสร้าง
  - Properties
  - References
  - Content References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties → เก็บข้อมูลเกี่ยวกับเกมหลังคอมไพล์แล้ว
  - References
  - Content References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References → เก็บข้อมูลว่าเกมนี้ใช้ library อะไรบ้าง
  - Content References
  - Content
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content) → เก็บ content (รูป, เพลง) ของเกม
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References → เกมนี้ใช้ content project ไหนบ้าง
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References
  - Game.ico →ไอคอน
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References
  - Game.ico
  - Game1.cs →นิยามเกม
  - GameThumbnail.png
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png →รูปแทนเกมในเครื่อง Xbox
  - Program1.cs
- FirstGameContent (Content)
  - References

ไฟล์ที่สร้างให้โดยอัตโนมัติ

- ดูช่อง Solution Explorer ทางด้านขวา
- FirstGame
  - Properties
  - References
  - Content References
  - Game.ico
  - Game1.cs
  - GameThumbnail.png
  - Program1.cs →ฟังก์ชัน main
- FirstGameContent (Content)
  - References

Program.cs

```
using System;

namespace FirstGame
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
}
```

Program.cs

- มีฟังก์ชัน main
  - ดังนั้นจึงเป็นส่วนที่นิยาม “โปรแกรมหลัก” ของเกม
- ใน main มีการสร้าง instance ของ Game1
- แล้วเรียกเมธอด run ของคลาส Game1
- เมธอด run คือตัวเกม
- คลาส Game1 จึงสำคัญที่สุด

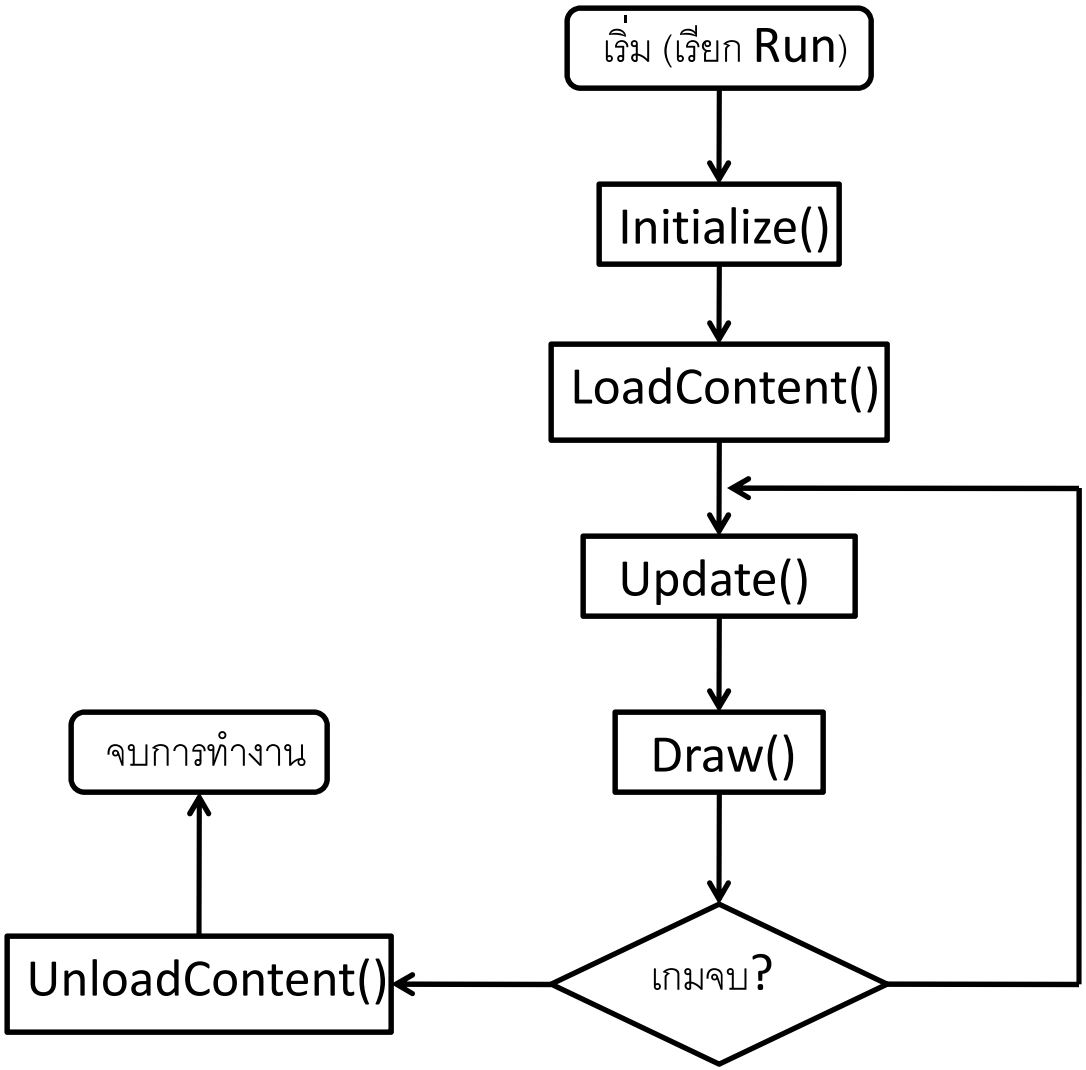
Game1.cs

- นิยามคลาส Game1 ซึ่งบรรจุรายละเอียดของเกมเอาไว้
- Game1 เป็น subclass ของ Microsoft.Xna.Framework.Game (คลาสที่แทนเกมของ XNA ทั้งหมด)

เมธอดของ Microsoft.XNA.Framework.Game

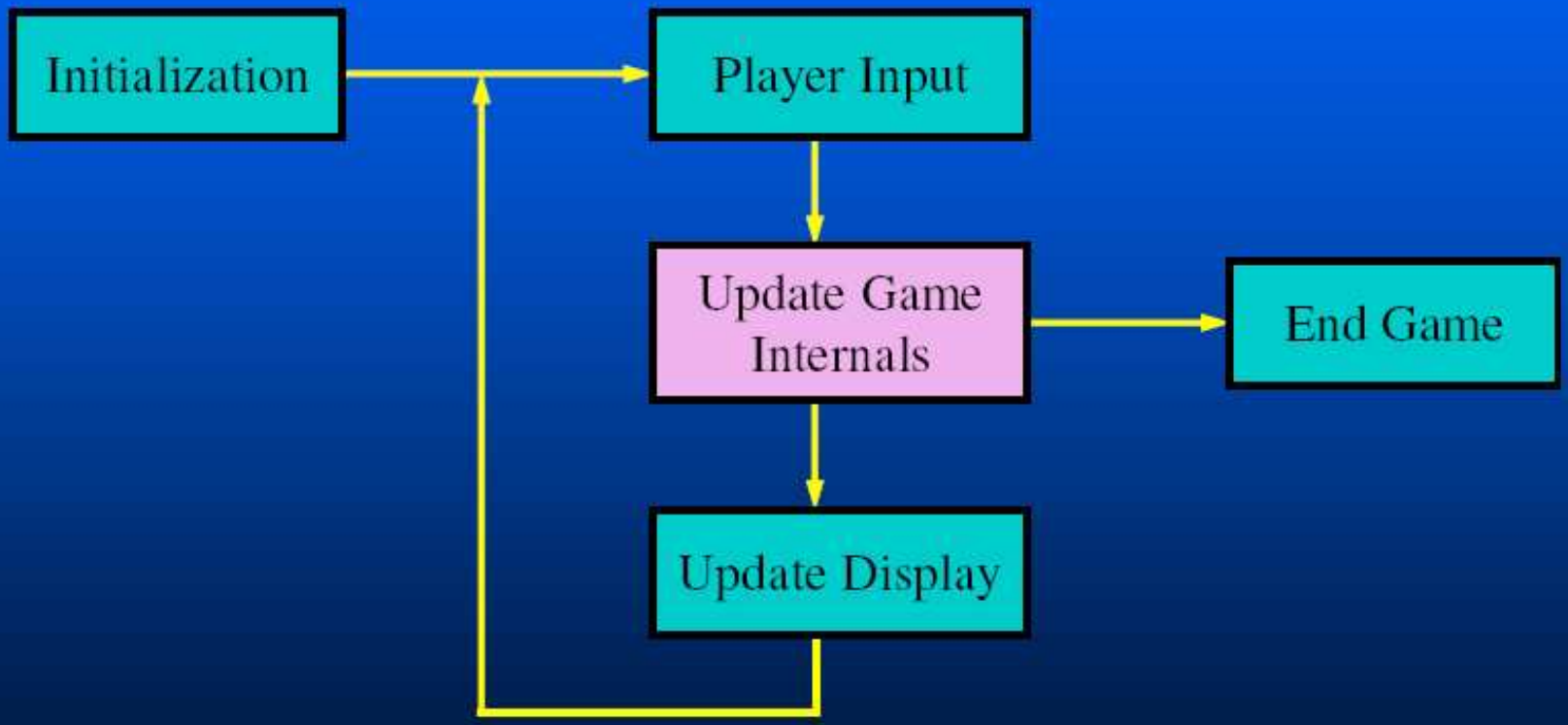
- void Initialize()
  - ใช้ตั้งค่าเริ่มต้นก่อนอ่าน content ขึ้นมา
- void LoadContent()
  - ใช้อ่าน content
- void UnloadContent()
  - ใช้ประมวลผลเพื่อนำ content ออกจากหน่วยความจำหลังเกมจบ
- void Update(GameTime gameTime)
  - ใช้ทำการเปลี่ยนแปลงสถานะภายในของเกม
  - เป็นเมธอดที่ logic ของเกมทำงาน
- void Draw(GameTime gameTime)
  - ใช้วาดภาพเกมออกทางหน้าจอ

Flowchart การทำงานของคลาส Game



Game Programming

The "Game Loop" (Main Event Loop) :



(This is the guts of every game)

เมธอดต่างๆ ใน Game1.cs

- Constructor

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

- GraphicsDeviceManager → จัดการหน้าจอ
- Content → จัดการ content ของเกม

เมธอดต่างๆ ใน Game1.cs

- Initialize()
  UnloadContent()
  - ไม่ได้ทำอะไรเลยนอกจากเรียก method เดียวกันของ superclass

- LoadContent()

```
protected override void LoadContent()
{
  spriteBatch = new SpriteBatch(GraphicsDevice);
}
```

สร้าง spriteBatch → object ที่เราจะใช้มันวาดรูป

เมธอดต่างๆ ใน Game1.cs

- Update()

```
protected override void Update(GameTime gameTime)
{
  // Allows the game to exit
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
      ButtonState.Pressed)
    this.Exit();

  // TODO: Add your update logic here

  base.Update(gameTime);
}
```

เมธอดต่างๆ ใน Game1.cs

- GamePad → ใช้รับ input จาก joystick ของ XBOX
- โค้ดบรรทัดแรกทำการเช็คว่าปุ่ม “back” ของผู้เล่นคนที่ 1 ถูกกดอยู่หรือไม่
- ถ้าใช่เรียก this.Exit() ซึ่งจะเป็นการจบเกม
- มิเช่นนั้นก็ไม่ทำอะไร

- เราจะเขียนเกมบน PC โดยใช้ Keyboard เป็นส่วนใหญ่
- ดังนั้นจะไม่ยุ่งกับ GamePad มากนัก

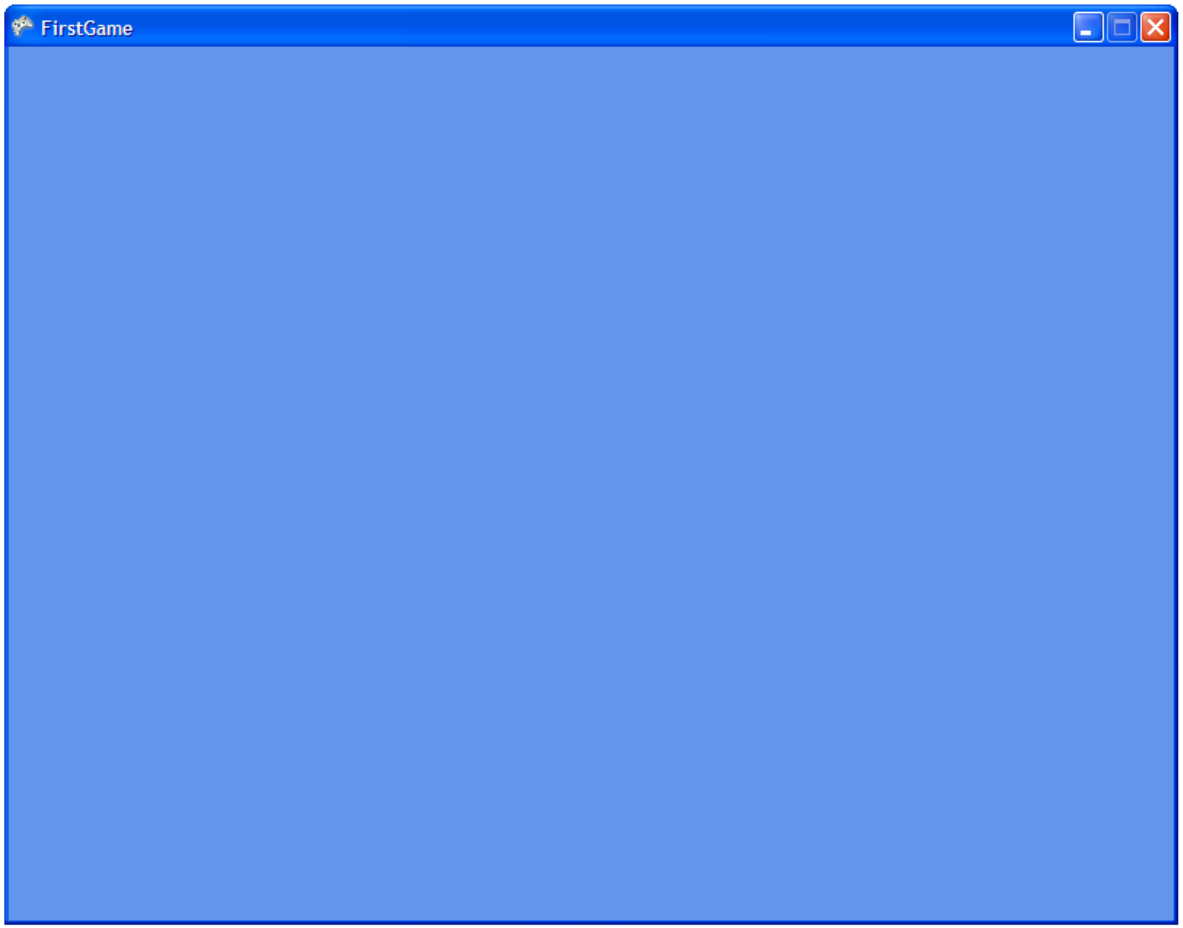
เมธอดต่างๆ ใน Game1.cs

- Draw()

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    base.Draw(gameTime);
}
```

- เคลียร์หน้าจอเป็นสี CornflowerBlue

ผลลัพธ์

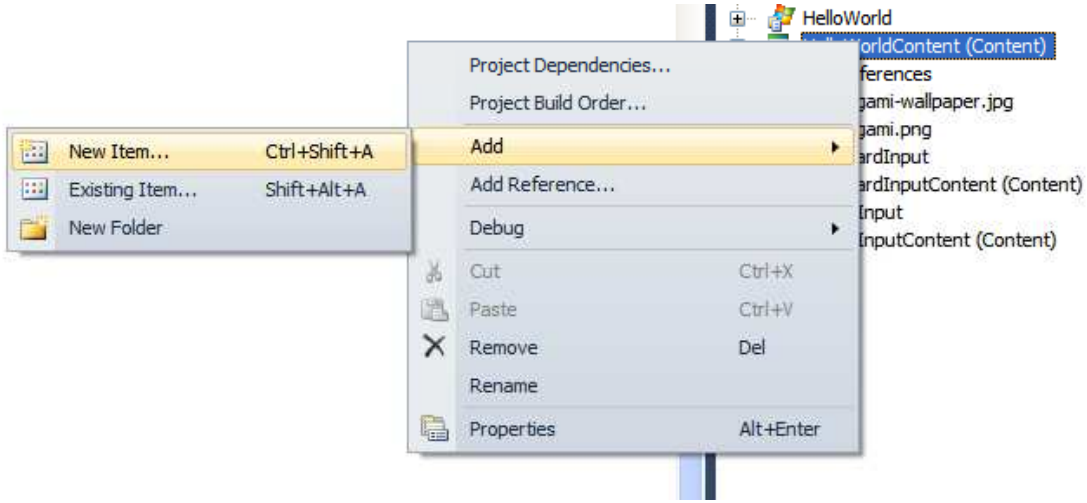


โปรแกรมแรก

โปรแกรมแรก

เพิ่มรูปเข้า Content

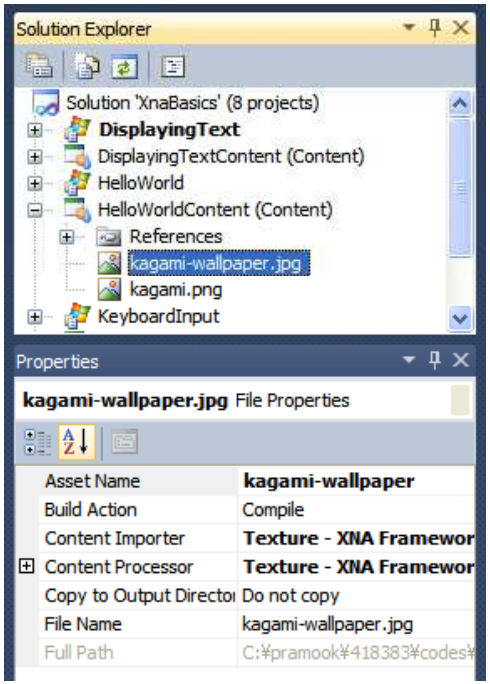
- ก่อนจะใช้รูปหรือเสียงในเกมได้ ต้องเพิ่มมันเข้าใส่ Content ก่อน
- คลิกขวาที่ HelloWorldContent แล้วเลือก Add → New Item…



- หลังจากนั้นให้เลือกรูปที่ต้องการเพิ่มเข้าเกม

เพิ่มรูปเข้า Content

- เมื่อลองขยาย Content จะมีชื่อรูปปรากฏอยู่



- ถ้าคลิกรูปแท็บ Properties จะบอก “Asset Name” ของรูป
- เราจะใช้ Asset Name นี้อ้างถึงรูปในภายหลัง

โหลดรูปเข้าหน่วยความจำ

- ก่อนจะนำรูปไปวาดได้ ต้องโหลดมันเข้าหน่วยความจำก่อน
  - ก่อนจะโหลดได้ รูปต้องอยู่ใน Content ก่อน

- รูปใน XNA ถูกเก็บด้วย object ชนิด Texture2D
- เรามีรูปที่จะใช้ 2 รูป: ตัวการ์ตูน, พื้นหลัง
- สร้าง field ชนิด Texture2D ในคลาสของเกม 2 field

```
public class HelloWorld : Microsoft.Xna.Framework.Game
{
  GraphicsDeviceManager graphics;
  SpriteBatch spriteBatch;

  Texture2D background;
  Texture2D sprite;
```

โหลดรูปเข้าหน่วยความจำ

- เวลาโหลดรูปเข้าหน่วยความจำ ให้ทำใน LoadContent()

```
protected override void LoadContent()
{
  spriteBatch = new SpriteBatch(GraphicsDevice);

  background = Content.Load<Texture2D>("kagami-wallpaper");
  sprite = Content.Load<Texture2D>("kagami");
}
```

Content.Load

- กำหนด type parameter เป็นชนิดของวัตถุที่เราต้องการอ่าน
  - ในที่นี้คือ Texture2D
- มี argument ตัวเดียว = Asset Name ของวัตถุที่จะโหลด
- คืนวัตถุชนิดที่เรากำหนดซึ่งตรงกับสื่อที่มี Asset Name ที่กำหนด

- เราสามรถใช้ Content.Load โหลด
  - รูป
  - เสียง
  - วัตถุที่เรานิยามเอง (การบรรยายครั้งที่ 6)

กำหนดขนาดของวินโดวส์

- เซตค่า
  - graphics.PreferredBackBufferWidth (ความกว้าง)
  - graphics.PreferredBackBufferHeight (ความสูง)

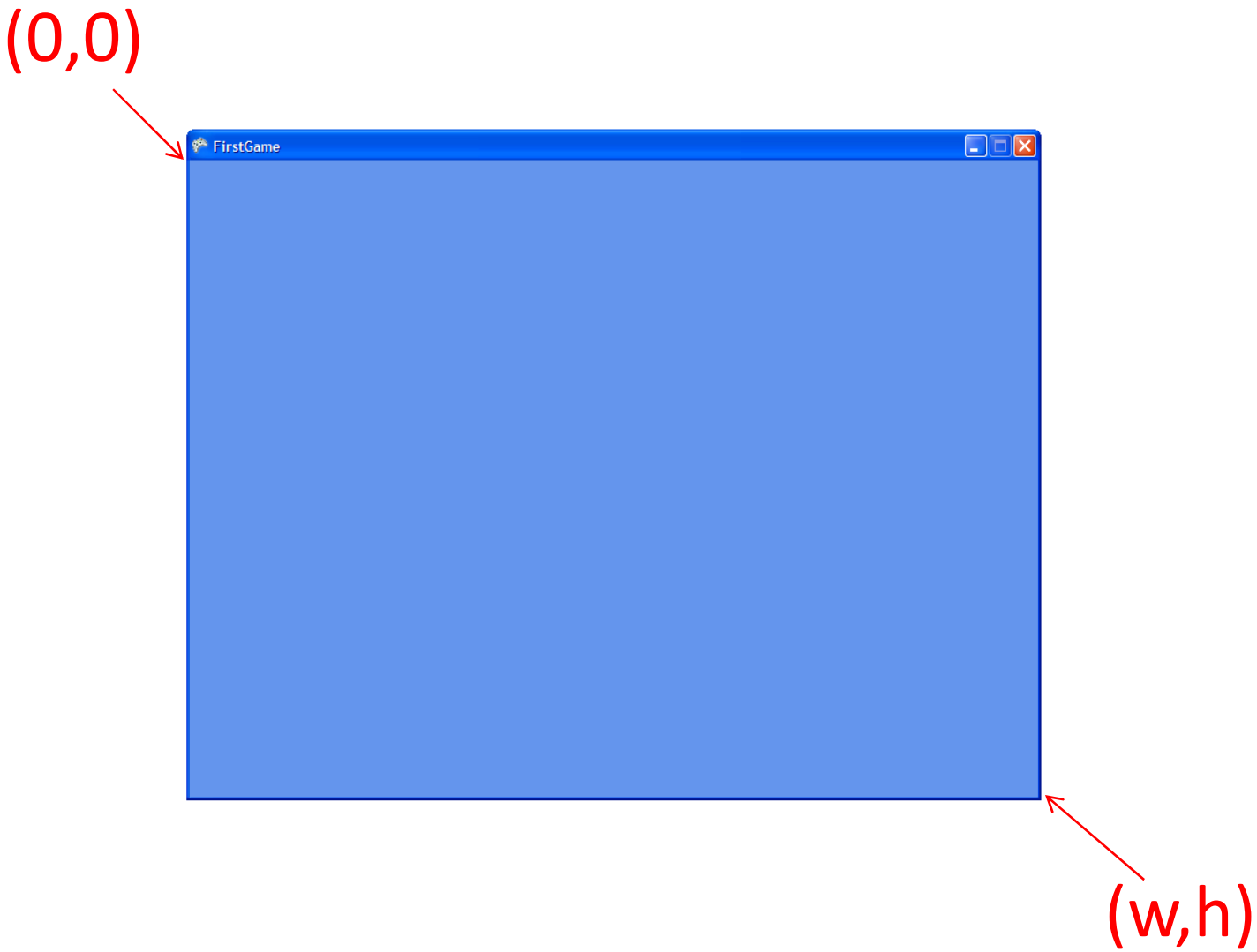
  ใน contructor

```
public HelloWorld()
{
  graphics = new GraphicsDeviceManager(this);

  // Setting the window's size.
  graphics.PreferredBackBufferWidth = 800;
  graphics.PreferredBackBufferHeight = 600;

  Content.RootDirectory = "Content";
}
```

ระบบ coordinate

- วินโดว์เป็นพื้นที่สี่เหลี่ยมผืนผ้า
- เราวัดตำแหน่งด้วยหน่วยพิกเซล
- (0,0) คือตำแหน่งมุมบนซ้ายของวินโดว์
- (w, h) คือตำแหน่งที่อยู่ที่มุมล่างขวาของวินโดว์
  เมื่อ w = ความกว้าง และ h = ความสูงของวินโดว์

ระบบ coordinate



ตำแหน่งของตัวการ์ตูน

- เราต้องการให้ตัวการ์ตูนเคลื่อนที่ตามเมาส์
- โดยให้**จุดศูนย์กลาง**ของตัวการ์ตูนอยู่ตรงกับตำแหน่งของเมาส์
- เวลาเราจะวาดตัวการ์ตูนเราจะต้องระบุตำแหน่ง**มุมบนซ้าย**ของตัวการ์ตูน
- เราสามารถเก็บตำแหน่งได้ด้วย object ชนิด Vector2
  - มี property ชื่อ X และ Y สำหรับเก็บพิกัดตามแกน X และ Y
- สร้าง field ชื่อ spritePosition ในคลาสของเกม

```
public class HelloWorld : Microsoft.Xna.Framework.Game
{
  GraphicsDeviceManager graphics;
  SpriteBatch spriteBatch;

  Texture2D background;
  Texture2D sprite;
  Vector2 spritePosition;
```

ทำให้ตัวการ์ตูนเคลื่อนตามเมาส์

- spritePosition ถือเป็นสถานะภายในของเกม
- ฉะนั้นเราเปลี่ยนมันในเมธอด Update

```
protected override void Update(GameTime gameTime)
{
  if ( ... )
    this.Exit();

  MouseState mouseState = Mouse.GetState();
  spritePosition.X = mouseState.X - sprite.Width / 2;
  spritePosition.Y = mouseState.Y - sprite.Height / 2;

  base.Update(gameTime);
}
```

Mouse

- เป็น object ที่สามารถให้ข้อมูลเกี่ยวกับเมาส์
- เมธอด GetState() คืน object ประเภท MouseState ซึ่งเก็บข้อมูลของสถานะปัจจุบันของเมาส์เอาไว้
- เราสามารถหาพิกัดแกน x และ y ของเมาส์ได้ โดยการเข้าถึง property X และ Y ของ MouseState

วาดรูป

- ในฟังก์ชัน Draw

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();
    spriteBatch.Draw(background, new Vector2(0, 0), Color.White);
    spriteBatch.Draw(sprite, spritePosition, Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

SpriteBatch

- ใช้สำหรับวาดรูป 2 มิติจาก Texture2D
- ก่อนจะใช้ต้องเรียกเมธอด Begin
- หลังจากวาดรูปเสร็จต้องเรียกเมธอด End
  - ถ้าไม่เรียกรูปจะไม่ถูกแสดงออกทางหน้าจอ
- วาดรูปด้วยเมธอด Draw
  - เมธอด Draw มี signature ทั้งหมด 7 แบบ
  - แต่ในโปรแกรมนี้เราจะใช้แค่แบบเดียว
- คำสั่ง draw ที่ถูกเรียกก่อน
  จะถูกคำสั่ง draw ที่ถูกเรียกทีหลังวาดทับ

SpriteBatch

```
spriteBatch.Draw(
    sprite, → Texture2D
    spritePosition,
    Color.White);
```

SpriteBatch

```
spriteBatch.Draw(
    sprite,
    spritePosition,
    Color.White);
```

→ ตำแหน่งมุมบนซ้าย

SpriteBatch

```
spriteBatch.Draw(
    sprite,
    spritePosition,
    Color.White);
```
→ สีที่ใช้ “คูณ” กับรูป

SpriteBatch

```
spriteBatch.Draw(
    sprite,
    spritePosition,
    Color.Red);
```

- ถ้าสั่งอย่างข้างบนจะได้รูปที่มีโทนสีแดง

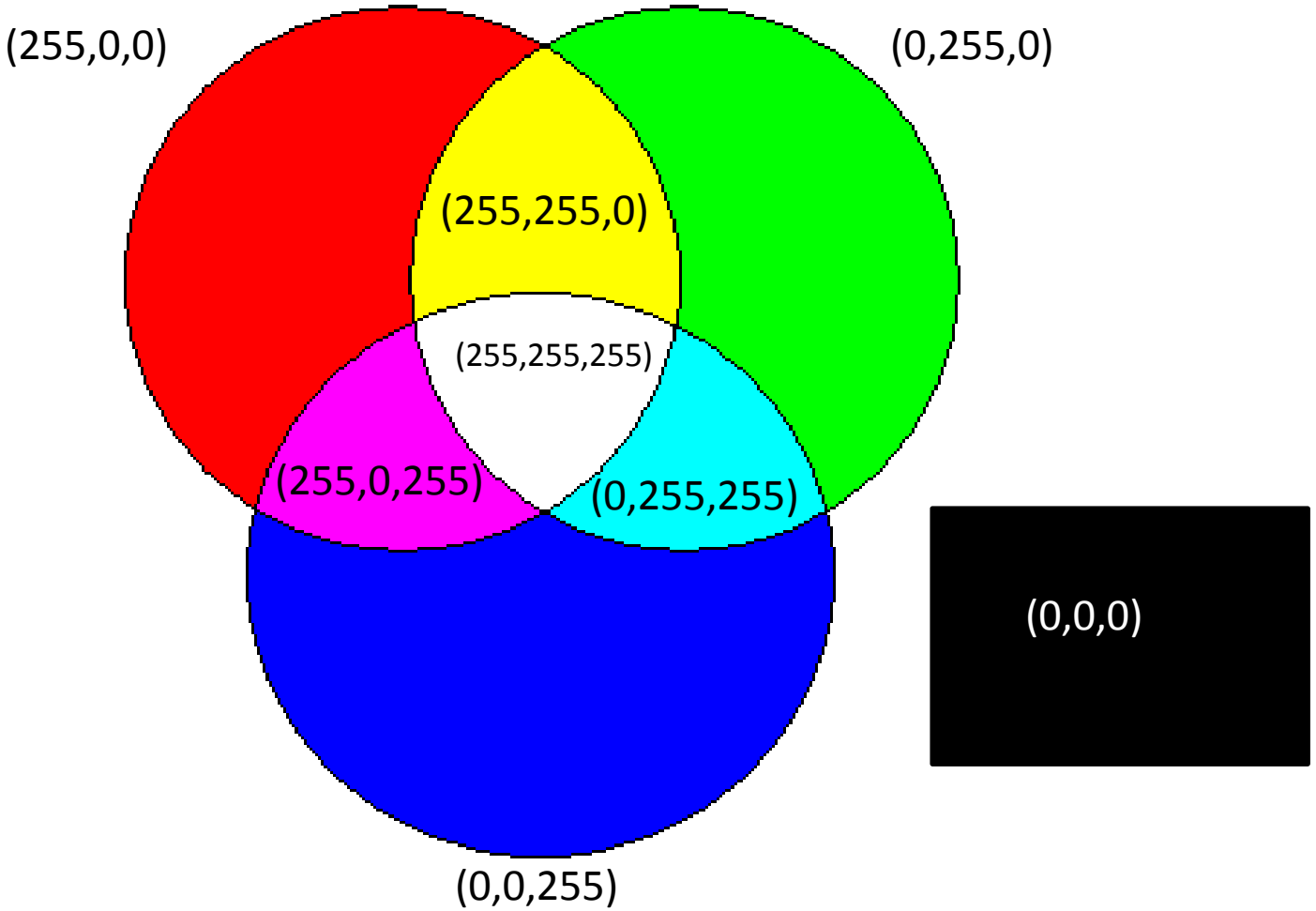
สี

- สี = tuple มีสมาชิกเป็นเลข 3 ตัว (R, G, B)
  - R บอกระดับความเข้มของแสงสีแดง
  - G บอกระดับความเข้มของแสงสีเขียว
  - B บอกระดับความเข้มของแสงสีน้ำเงิน
- เลขแต่ละตัวมีค่าตั้งแต่ 0 ถึง $2^{\text{จำนวนบิตที่ใช้เก็บความเข้มแต่ละสี}}-1$
  - ส่วนใหญ่เราจะใช้พื้นที่ 32 บิตเก็บ 1 พิกเซล
  - แต่ละสีได้ 8 บิต
  - ดังนั้นค่าสูงสุดคือ 255

Trichromatic Theory of Vision

- สีที่มนุษย์มองเห็นแบ่งออกเป็นสามส่วน
  - แดง เขียว น้ำเงิน
  - ประสาทสัมผัสของมนุษย์ของแต่ละสีเป็นอิสระจากกัน
  - สีอื่นๆ เกิดจาก การนำสีทั้งสามนี้มาประกอบกัน
- หลักฐาน
  - เซลล์โคนในเรตินามีสามชนิด
  - แต่ละชนิดไวต่อ สีแดง สีเขียว สีน้ำเงิน ตามลำดับ

สีหลักๆ



Color

- คลาสที่ใช้เก็บสี
- เราสามารถสร้างสีด้วยการกำหนดค่า RGB ได้

```
Color color1 = new Color(128, 0, 255);
```

- หรือกำหนดด้วยทศนิยมโดยที่ 0 หมายถึงไม่มีความเข้ม และ 1 หมายถึงเข้มที่สุด

```
Color color2 = new Color(0.5f, 0, 1);
```

- นอกจากนี้ยังมี static field ที่มีค่าเป็นสีที่เราใช้บ่อยๆ เช่น

```
Color.Green, Color.Blue, Color.Lavender, ฯลฯ
```

โปรแกรมเลื่อนตัวการ์ตูนด้วยคีย์บอร์ด

โปรแกรมเลื่อนตัวการ์ตูนด้วย keyboard

Keyboard

- คลาสสำหรับอ่านสถานะของคีย์บอร์ด
- เมธอดสำคัญ: GetState()
  - คืน object ประเภท KeyboardState

KeyboardState

- เก็บสถานะของปุ่มต่างๆ ของ keyboard เอาไว้
- เมธอดสำคัญ:
  - IsKeyDown(Keys key)
    - ตรวจว่าปุ่มถูกกดอยู่หรือไม่
  - IsKeyUp(Keys key)
    - ตรวจว่าปุ่มถูกปล่อยอยู่หรือไม่

Keys

- enum ที่ใช้แทนปุ่มบนคีย์บอร์ด เช่น
  - Keys.A → ปุ่ม A
  - Keys.Space → ปุ่ม space bar
  - Keys.Right → ปุ่มลูกศรขวา

เช็คว่าปุ่มถูกกดหรือไม่

```
KeyboardState keyboardState = Keyboard.GetState();
if (keyboardState.IsKeyDown(Keys.Left))
{
  ... Do something ...
}
```

โปรแกรมเลื่อนตัวการ์ตูนด้วยคีย์บอรด์

- มีฟิลด์ spritePosition แทนตำแหน่ง**มุมบนซ้าย**ของตัวการ์ตูนเหมือนเดิม
- มีฟิลด์
  - xVelocity → แทนความเร็วในการเคลื่อนที่ตามแนวนอน
  - yVelocity → แทนความเร็วในการเคลื่อนที่ตามแนวตั้ง
  - ทั้งคู่มีขนาดเท่ากับ 5 แต่คนละทิศทาง

```
Vector2 xVelocity = new Vector2(5, 0);
Vector2 yVelocity = new Vector2(0, 5);
```

การเลื่อนตัวการ์ตูน

- บวกความเร็วเข้ากับตำแหน่งถ้าผู้ใช้กดปุ่มลูกศรทิศทาง

```
protected override void Update(GameTime gameTime)
{
    เช็คว่าผู้ใช้กดปุ่ม “back” เพื่อเลิกการทำงานหรือไม่

    KeyboardState keyboardState = Keyboard.GetState();

    if (keyboardState.IsKeyDown(Keys.Left))
        spritePosition -= xVelocity;
    if (keyboardState.IsKeyDown(Keys.Right))
        spritePosition += xVelocity;
    if (keyboardState.IsKeyDown(Keys.Down))
        spritePosition += yVelocity;
    if (keyboardState.IsKeyDown(Keys.Up))
        spritePosition -= yVelocity;

    เช็คว่าตัวการ์ตูนตกขอบหรือไม่

    base.Update(gameTime);
}
```

เช็คว่าตัวการ์ตูนตกขอบหรือไม่

```
int screenWidth = graphics.PreferredBackBufferWidth;
int screenHeight = graphics.PreferredBackBufferHeight;

if (spritePosition.X < 0)
    spritePosition.X = 0;
if (spritePosition.X + sprite.Width > screenWidth)
    spritePosition.X =  screenWidth - sprite.Width;
if (spritePosition.Y < 0)
    spritePosition.Y = 0;
if (spritePosition.Y + sprite.Height > screenHeight)
    spritePosition.Y = screenHeight - sprite.Height;
```

โปรแกรมลากตัวการ์ตูนด้วยเมาส์

โปรแกรมลากตัวการ์ตูนด้วยเมาส์

Mouse

- คลาสสำหรับอ่านสถานะของเมาส์
- เมธอดสำคัญ: GetState()
  - คืน object ประเภท MouseState

MouseState

- สถานะปัจจุบันของเมาส์
- Property สำคัญ
  - LeftButton → สถานะของปุ่มซ้าย
  - RightButton → สถานะของปุ่มขวา
  - MiddleButton → สถานะของปุ่มกลาง
  - X → ตำแหน่งแนวตั้ง
  - Y → ตำแหน่งแนวนอน

ButtonState

- Enum ที่ใช้แทนสถานะของปุ่มเมาส์
- มีสมาชิกสองตัว
  - Pressed → ถูกกดอยู่
  - Released → ไม่ถูกกดอยู่

ทำลูกศรเมาส์

- มีฟิลด์สำหรับเก็บตำแหน่งลูกศรและภาพของลูกศร

```
Texture2D cursor;
Vector2 cursorPosition;
```

- ใน Update() ให้เปลี่ยนตำแหน่งลูกศร

```
protected override void Update(GameTime gameTime)
{
    การประมวลผลอย่างอื่น

    MouseState mouseState = Mouse.GetState();
    cursorPosition = new Vector2(mouseState.X, mouseState.Y);

    การประมวลผลอย่างอื่น
}
```

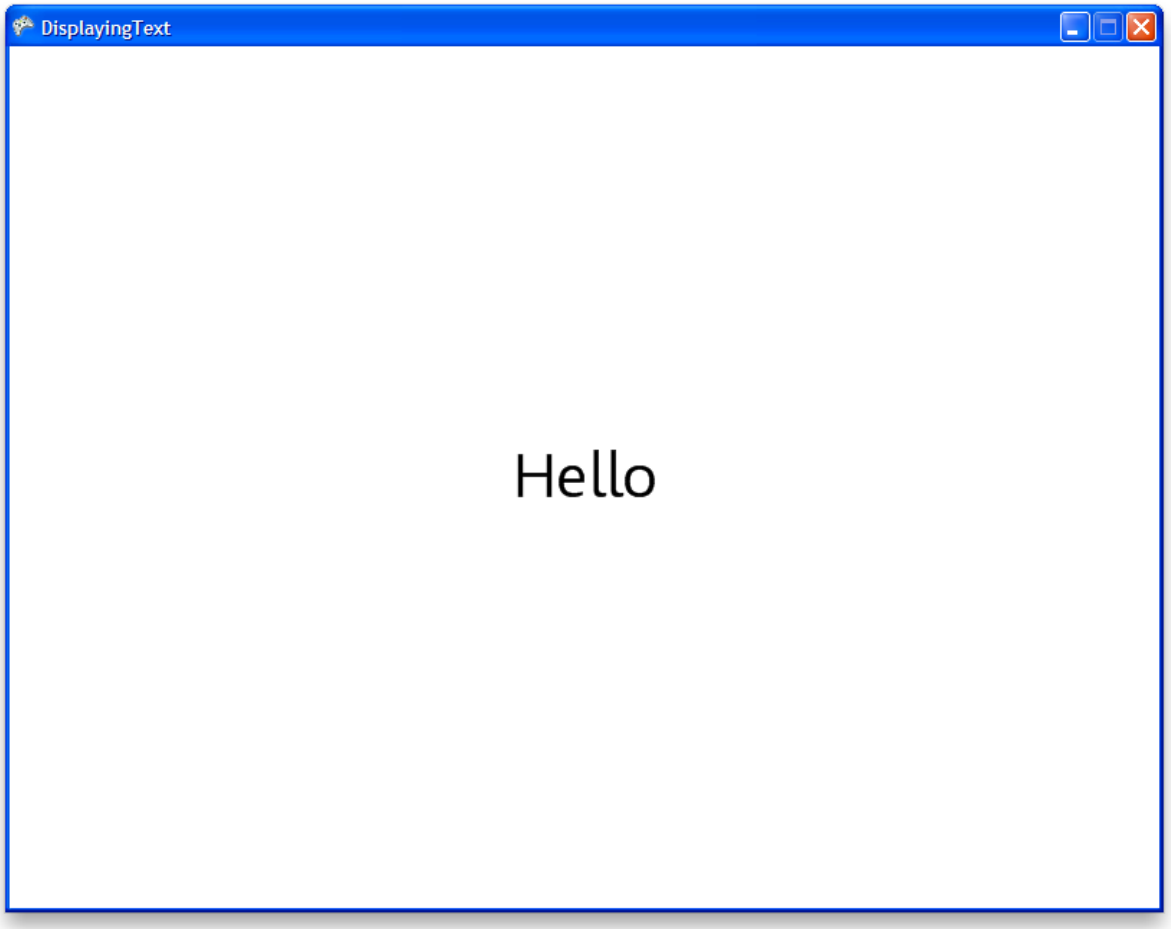
ทำลูกศรเมาส์

- ใน Draw() ก็ให้วาดลูกศร
- ต้องวาดอย่างอื่นให้เสร็จก่อน เพราะตัวที่วาดทีหลังจะทับตัวที่ถูกวาดก่อน

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin();
    // วาดอย่างอื่นก่อน //
    spriteBatch.Draw(cursor, cursorPosition, Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

โปรแกรมวาดข้อความ

โปรแกรมวาดข้อความ

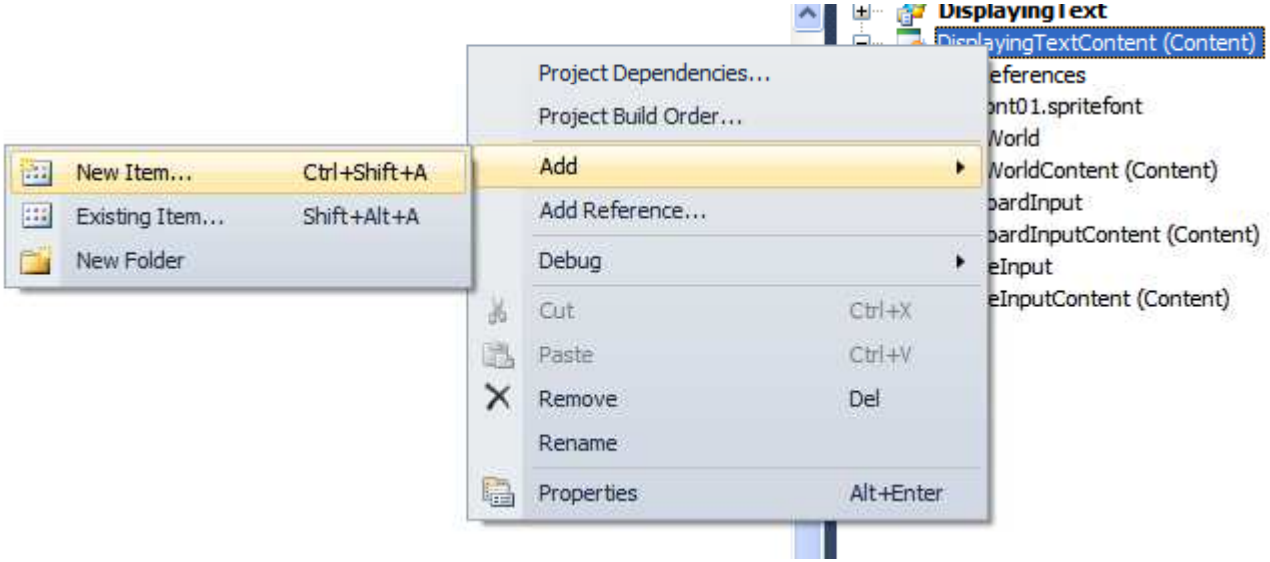


การวาดข้อความใน XNA

1. สร้าง Sprite Font ใน Content ของเกม
2. Load Sprite Font ใน Load Content
3. วาดข้อความด้วย SpriteBatch.DrawString

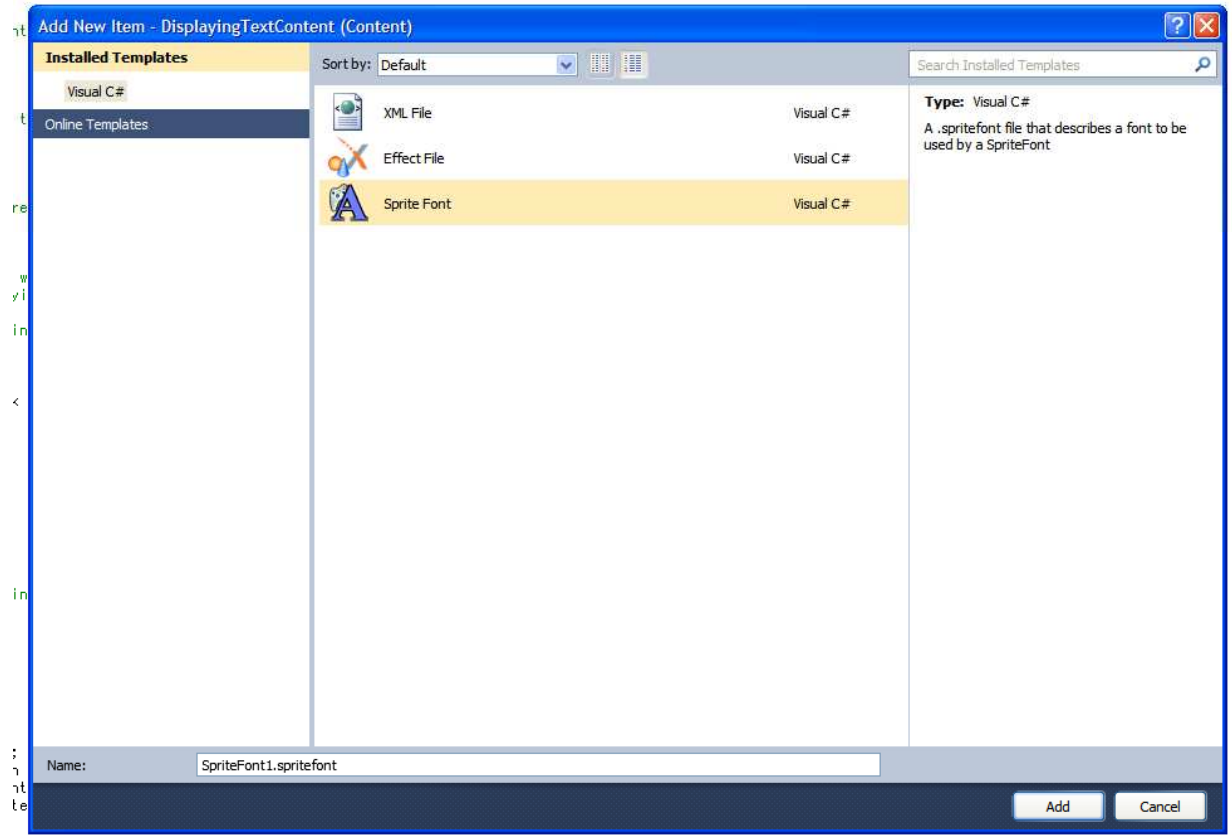
สร้าง Sprite Font ใน Content

- คลิกขวาที่โฟลเดอร์ DisplayingTextContent →Add → New Item



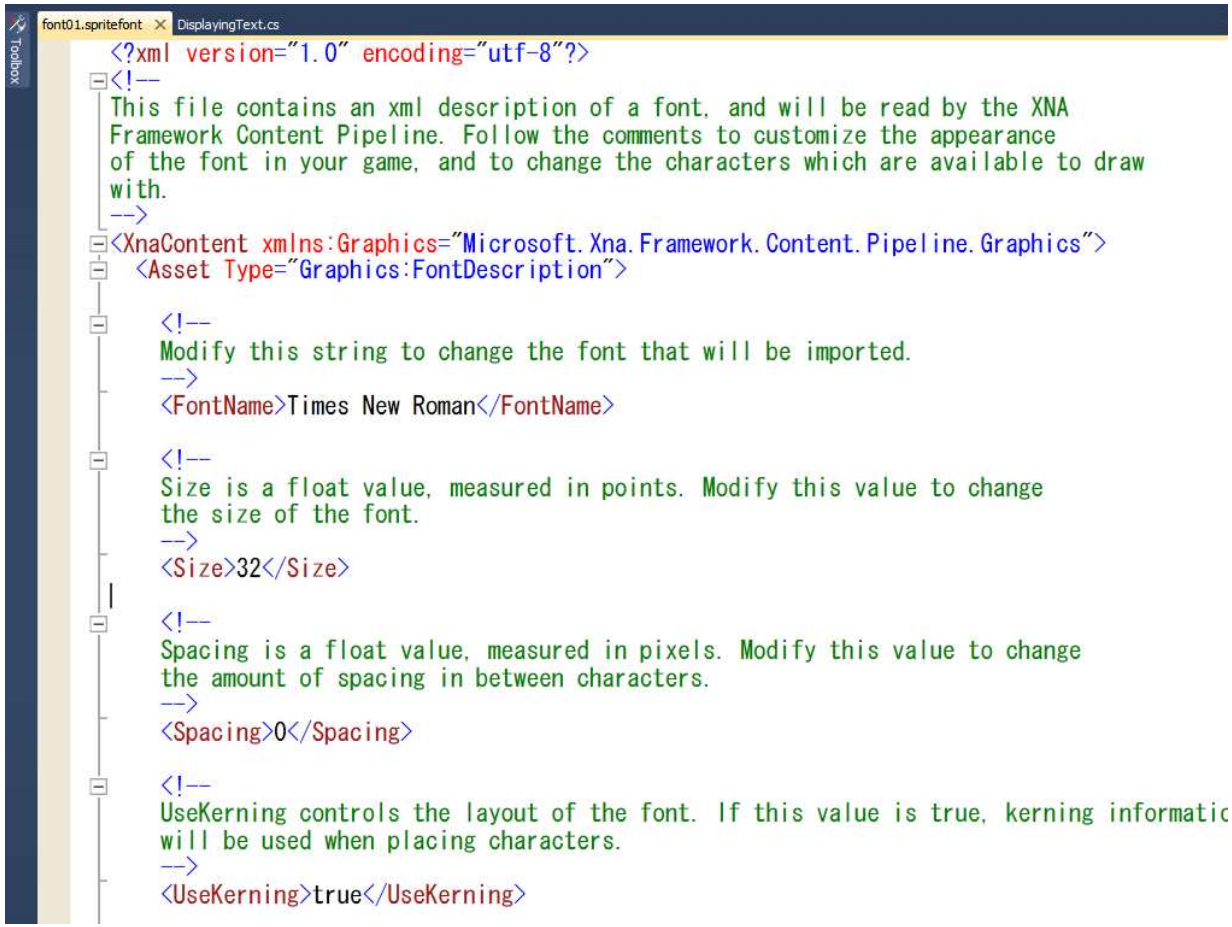
สร้าง Sprite Font ใน Content

- เลือก Sprite Font



สร้าง Sprite Font

- เปิดไฟล์ sprite font ที่สร้างใหม่แล้วแก้ไข
  - ชื่อฟอนต์
  - ขนาด
  - ฯลฯ



Load Sprite Font ใน LoadContent()

- สร้างฟิลด์ชนิด SpriteFont

```
SpriteFont font01;
```

- ใช้ Content.Load<SpriteFont>(...) ในการโหลดมัน

```
protected override void LoadContent()
{
    // โหลดอย่างอื่น //
    font01 = Content.Load<SpriteFont>("font01");
}
```

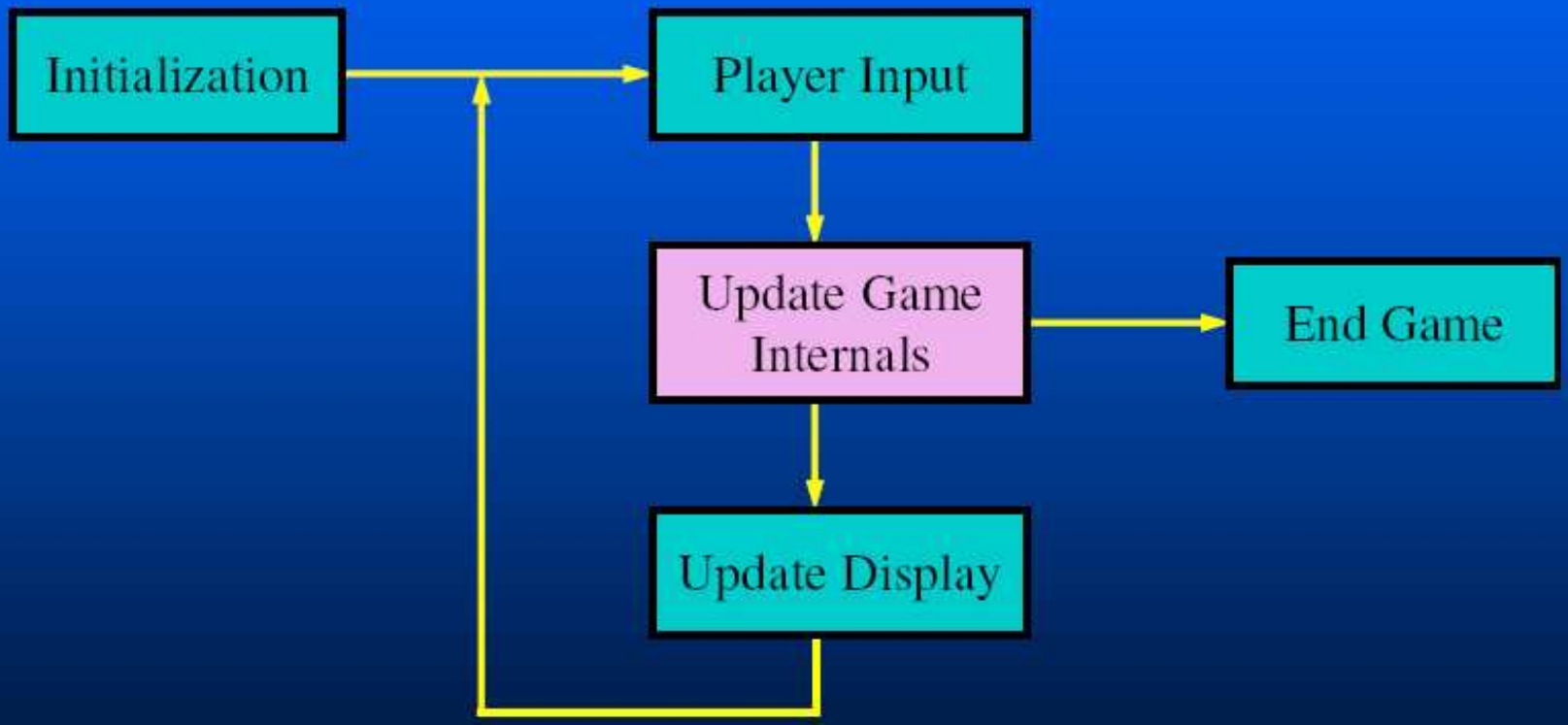
วาดข้อความด้วย SpriteBatch.DrawString

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);

    spriteBatch.Begin();

    // Draw "Hello" at the center of the screen.
    Vector2 helloSize = font01.MeasureString("Hello");
    float centerX = (graphics.PreferredBackBufferWidth - helloSize.X) / 2;
    float centerY = (graphics.PreferredBackBufferHeight - helloSize.Y) / 2;
    Vector2 centerPosition = new Vector2(centerX, centerY);

    spriteBatch.DrawString(
        font01,
        "Hello",
        centerPosition,
        Color.Black);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

โครงสร้างของเกม

Game Programming

The "Game Loop" (Main Event Loop) :



(This is the guts of every game)

แต่มันไม่ง่ายอย่างนั้น!

- เกมมีหน้าจอหลายหน้าจอ
  - ภาพยนตร์เปิด
  - หน้าไตเติ้ล
  - หน้าเล่นเกม
  - หน้าต่อสู้
  - หน้าแผนที่
  - หน้าเกมโอเวอร์
  - หน้าสำหรับเซฟเกม
  - ฯลฯ

ทำไม

- หน้าจอแต่ละหน้ามีวิธีการ
  - วาดสิ่งที่มันต้องการแสดงให้ผู้ใช้เห็นเอง
  - จัดการข้อมูลของมันเอง
  - จัดการปฏิสัมพันธ์กับผู้ใช้เอง

แล้วเราจะจัดการกับมันอย่างไร?

- ไอเดียที่ไม่ดี
  - สร้างตัวแปรตัวหนึ่งเพื่อเก็บไว้ว่าตอนนี้เราอยู่ที่ “หน้าจอ” ไหน
  - ให้ตัวแปรนี้เป็นตัวบอกว่าเราจะ
    - วาดรูปอย่างไร
    - จัดการข้อมูลภายในเกมอย่างไร และ
    - จัดการกับปฏิสัมพันธ์กับผู้ใช้อย่างไร
  - ดังนั้น ในส่วนต่างๆ ของเกมลูป จะมีคำสั่ง if ขนาดใหญ่อยู่หนึ่งคำสั่ง
    - อย่างน้อยมีคำสั่ง if สามคำสั่งในหนึ่งโปรแกรม

แล้วเราจะจัดการกับมันอย่างไร?

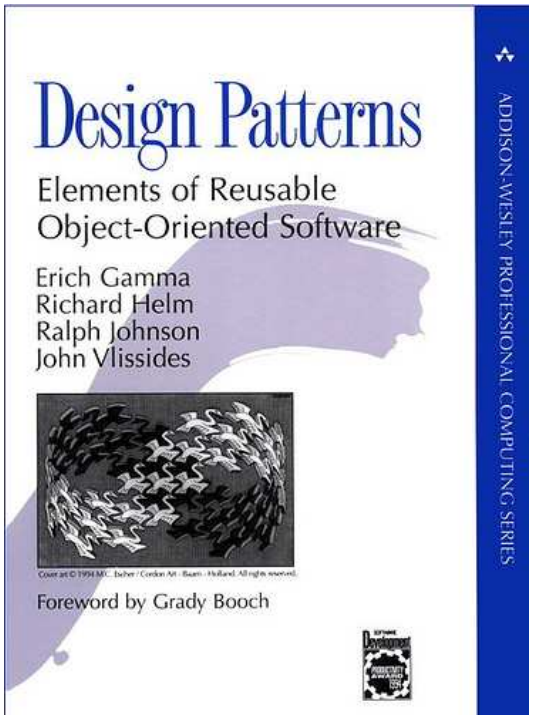
- ทำไมมันถึงเป็นไอเดียที่ไม่ดี
  - คำสั่ง **if** ที่ว่าจะมีขนาดใหญ่ถ้าโปรแกรมมีหลายๆ หน้าจอ
  - โค้ดที่กำหนดการทำงานของหน้าจอเดียวกันถูกแยกออกไปอยู่ในคำสั่ง **if** หลายๆ คำสั่ง
  - สมมติว่าตอนเพิ่มหน้าจอใหม่ เราลืมเอาไปดูคำสั่ง **if** สักหนึ่งที่ล่ะ? จะเกิดอะไรขึ้น?

Design Patterns

- “คือ บันทึกวิธีการแก้ปัญหาเกี่ยวกับการออกแบบในสาขาวิชาใดวิชาหนึ่ง” (Wikipedia)
- ในทางการออกแบบซอฟต์แวร์ มันคือวิธีการแก้ปัญหาการออกแบบซอฟต์แวร์ที่ได้รับการทดสอบจนเป็นที่ยอมรับ
- คุณเคยเรียน pattern ไหนมาจากวิชา Java หรือ SA กันบ้าง?

The Gang of Four

- หนังสือมาตรฐานเกี่ยวกับ Design Patterns
- โปรแกรมเมอร์ทุกคนควรอ่าน ไม่งั้นเสียชาติเกิด



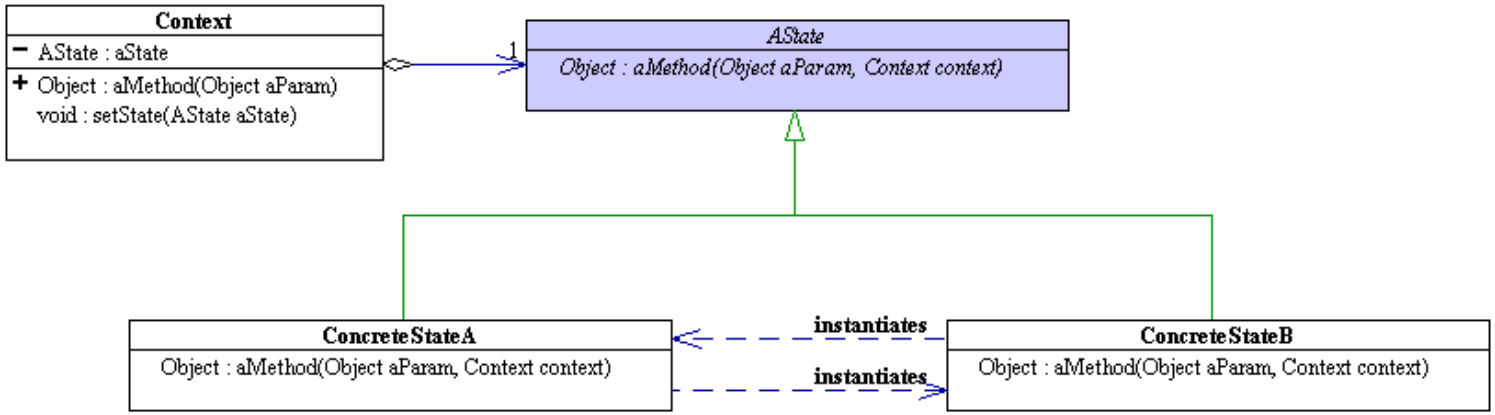
แล้วมันเกี่ยวอะไรกันกับการเขียนเกม?

- ปัญหาการมี “หลายหน้าจอ” ไม่ใช่ปัญหาใหม่
- วิธีการแก้ปัญหานี้ถูกรวมไว้เป็น design pattern แล้ว
- เรียกว่า “State” design pattern

State Design Pattern

- “มีไว้ทำให้ออบเจกต์เปลี่ยนแปลงพฤติกรรมของตัวเองเวลาสถานะภายในของมันเปลี่ยน จนดูเหมือนว่าคลาสของออบเจกต์นั้นถูกเปลี่ยนกลางคัน”
- ฟังแล้วคุ้นๆ ไหม?
- ลองแทนคำว่า:
  - “ออบเจกต์” ด้วย “เกม”
  - “สถานะภายใน” ด้วย “หน้าจอ”
  - “คลาส” with “รูปแบบการติดต่อกับผู้ใช้”

The State Design Pattern



State Design Pattern

- Context (บริบท)
  - ผู้ใช้ทำการติดต่อกับคลาสนี้
  - ภายในเก็บ instance ของ ConcreteState ซึ่งเป็นซับคลาสของ State เอาไว้ เจ้าตัว instance นี่แหละคือสถานะภายในที่สามารถเปลี่ยนแปลงไปได้
- State
  - มี interface ที่ครอบคลุมพฤติกรรมที่เปลี่ยนแปลงไปตามสถานะภายใน
- ConcreteState
  - มีโค้ดที่ทำให้เกิดพฤติกรรมที่ตรงกับสถานะภายในหนึ่งๆ

เอามันมาใช้กับเกม

- “Context” → “Game”
  - สืบเชื้อสายมาจาก Microsoft.Xna.Framework.Game
- “State” → “Screen”
- ในคลาส Game มีฟิลด์ “currentScreen” สำหรับเก็บ screen ปัจจุบันของเกม
- ถ้าต้องการเปลี่ยน screen ก็แค่เปลี่ยนค่าตัวแปร currentScreen

คลาส Screen

- มี method ที่ต่างๆ ที่สำคัญของ Game
  - Initialize
  - Update(GameTime gameTime)
  - LoadContent
  - UnloadContent
  - Draw(GameTime gameTime)
  - Update(GameTime gameTime)
- Method เหล่านี้ส่วนใหญ่เป็น abstract method เนื่องจากเราต้องการให้ผู้ใช้ Screen ไป subclass มัน

คลาส Screen

- มี field สำหรับเก็บข้อมูลที่จำเป็น
  - name สำหรับเก็บชื่อ
  - game สำหรับเก็บ instance ของ Game ที่ screen นี้เป็นสมาชิก
- มี property สำหรับ field ทั้งสองข้างต้น
  - Name
  - Game
  - ทั้งสอง property นี้เป็นแบบ read-only (มีแต่ get)

คลาส Screen

- มี method อำนวยความสะดวกอื่นๆ
  - void SwitchedIntoFrom(string sceneName)
    - ถูกเรียกเมื่อ screen ถูกเปลี่ยนมายัง screen นี้
    - รับชื่อของ screen เดิมเป็น argument
  - void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    - วาดรูปลงบนหน้าจอด้วย SpriteBatch ที่ได้รับ
    - เป็น abstract method ที่ผู้ใช้ต้อง override
    - ถูกเรียกโดย Draw(GameTime gameTime) ซึ่งจะเอา SpriteBatch ของ Game ที่ screen นี้เป็นส่วนประกอบอยู่มาใช้

คลาส Game

- สืบเชื้อสายมาจาก Microsoft.Xna.Framework.Game
- เวลาเขียนเกมในวิชานี้เราจะสืบเชื้อสายเกมมาจากคลาสนี้ จะไม่ใช้ Microsoft.Xna.Frame.Game โดยตรงอีก

- มี field
  - currentScreen สำหรับเก็บ screen ปัจจุบัน
  - screens เป็น Dictionary ที่ map ชื่อ screen ไปยัง screen ต่างๆ ในเกม

คลาส Game

```
private Screen currentScreen;
public Screen CurrentScreen
{
      get { return currentScreen; }
}
```

Dictionary<string, Screen> screens;

คลาส Game

- เวลาเพิ่ม screen ใส่เกม ให้เรียก method
  void AddScreen(Screen screen)

```
public void AddScreen(Screen screen)
{
    screens[screen.Name] = screen;
}
```

คลาส Game

- เวลาเปลี่ยน screen ให้เรียก method
  void SwitchScreen(string name)

```
public void SwitchScreen(string name)
{
    var newScreen = GetScreenByName(name);
    var oldScreen = CurrentScreen;
    currentScreen = newScreen;
    if (oldScreen != null)
        currentScreen.SwitchedIntoFrom(oldScreen.Name);
    else
        currentScreen.SwitchedIntoFrom("null");
}
```

คลาส Game

- Screen GetScreenByName(string name)
  มีไว้สำหรับดึง screen ที่มีชื่อที่กำหนดออกมา

```
public Screen GetScreenByName(string name)
{
    if (screens.ContainsKey(name))
        return screens[name];
    else
        throw new InvalidOperationException(
            "Screen by name '" + name + "' not found.");
}
```

คลาส Game

- override method ของ Xna.Framework.Game โดยในเมธอดนั้นจะมีการเรียก method ที่ตรงกันของ Screen

```
protected override void Initialize()
{
        foreach (Screen screen in screens.Values)
                screen.Initialize();
        base.Initialize();
}
```

คลาส Game

```
protected override void LoadContent()
{
    base.LoadContent();
    spriteBatch = new SpriteBatch(GraphicsDevice);
    foreach (Screen screen in screens.Values)
        screen.LoadContent();
}

protected override void UnloadContent()
{
    foreach (Screen screen in screens.Values)
        screen.UnloadContent();
    base.UnloadContent();
}
```

คลาส Game

```
protected override void Update(GameTime gameTime)
{
    base.Update(gameTime);
    currentScreen.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    currentScreen.Draw(gameTime);
}
```

กรณีศึกษา: Tetris

- เกม Tetris อย่างง่ายอาจมี screen อยู่ 4 screen
  - Title Screen – หน้าจอไตเติ้ล
  - Play Screen – เวลาเล่นเกมที่ผู้ใช้สามารถบังคับเกมได้
  - Full Row Screen – แสดงผลเวลาแถวแถวหนึ่งเติม ให้มันกระพริบแล้วหายไป
  - Game Over Screen – แสดงข้อความ Game Over

เกมในฐานะ State Machine

- เกมมีหลาย “หน้าจอ”
- ในเวลาหนึ่งๆ มีเพียงหน้าจอเดียวเท่านั้นที่ถูกแสดงผล (active)
- มีการเปลี่ยนหน้าจอหนึ่งไปยังอีกหน้าจอหนึ่ง ซึ่งการเปลี่ยนนี้ถูกควบคุมโดย “ปัจจัยภายนอก” (ผู้ใช้และเวลาที่ผ่านไป)
- เวลาเขียนเกมเราจะสร้าง subclass ของ Screen หนึ่ง subclass สำหรับหน้าจอแต่ละแบบ
- เวลามีการเปลี่ยนแปลงหน้าจอ เราก็จะเรียก SwitchScreen

- เราสามารถเขียนการเปลี่ยนสถานะของเกมด้วยแผนภาพที่เรียกว่า “transition diagram”

Transition Diagram ของเกม Tetris