418383: การเขียนโปรแกรมเกม
เขียนเกม Breakout

ประมุข ขันเงิน

pramook@gmail.com

Breakout

- เผยแพร่ครั้งแรกในเครื่อง Arcade ในปี 1976 โดยบริษัท Atari
- ออกแบบโดย Nolan Bushnell และ Steve Bristow
- ได้รับแรงบรรดาลใจมาจากเกม Pong

Screenshot

วิดีโอ

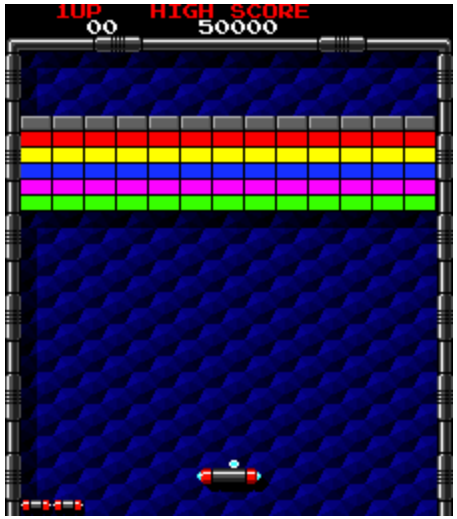
- http://www.youtube.com/watch?v=JRAPnuwnpRs

กติกา

- มีบล็อกอยู่ 8 แถว
- ผู้เล่นบังคับแป้นที่อยู่ด้านล่างของจอให้เคลื่อนไปทางซ้ายขวา
- เมื่อลูกบอลโดนบล็อก มันจะเด้งกลับและบล็อกจะถูกทำลาย
- เมื่อลูกบอลโดนแป้นมันจะเด้งกลับขึ้นข้างบน
- ผู้เล่นต้องเลี้ยงลูกบอลเพื่อทำลายบล็อกให้มากที่สุด โดยไม่ให้ลูกบอลตกลงไปใต้แป้น
- ถ้าลูกบอลตกลงไปใต้แป้น ผู้เล่นจะเสียตาเล่น
- ผู้เล่นมีตาเล่นเริ่มต้นอยู่สามตา ถ้าตาหมดจะ game over

Arkanoid

- เกมสร้างเลียนแบบ Breakout
- เผยแพร่โดยบริษัท Taito ในปี 1986
- ประสบความสำเร็จมากและมี Arkanoid clone ออกมาหลายตัว
- ฟีเจอร์เพิ่มเติม
  - มีหลายด่าน
  - มีไอเทมให้เก็บ
  - มียานอวกาศศัตรูให้ยิงทำลาย



วิดีโอ

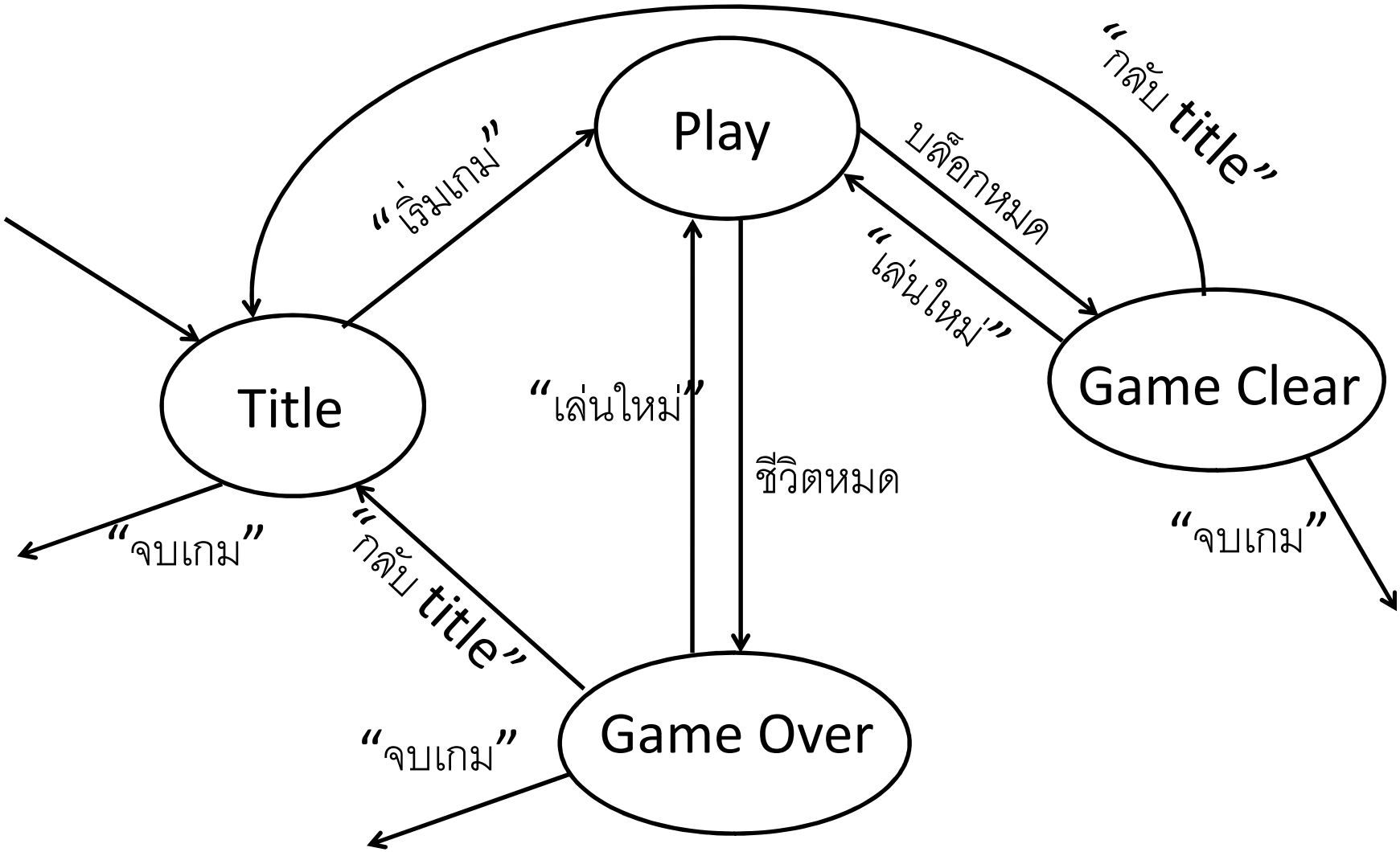
- http://www.youtube.com/watch?v=44rceRqY8_k

BREAKOUT IMPLEMENTATION

เกมที่เราจะเขียน

- Arkanoid clone
- มีไอเทมต่อไปนี้ให้เก็บ
  - ทำให้แป้นยาวขึ้น
  - ทำให้แป้นดูดลูกบอล แล้วให้ผู้ใช้ปล่อยลูกบอลได้ทีหลัง
- มีเพียงแค่ด่านเดียว
- ไม่มียานของศัตรู

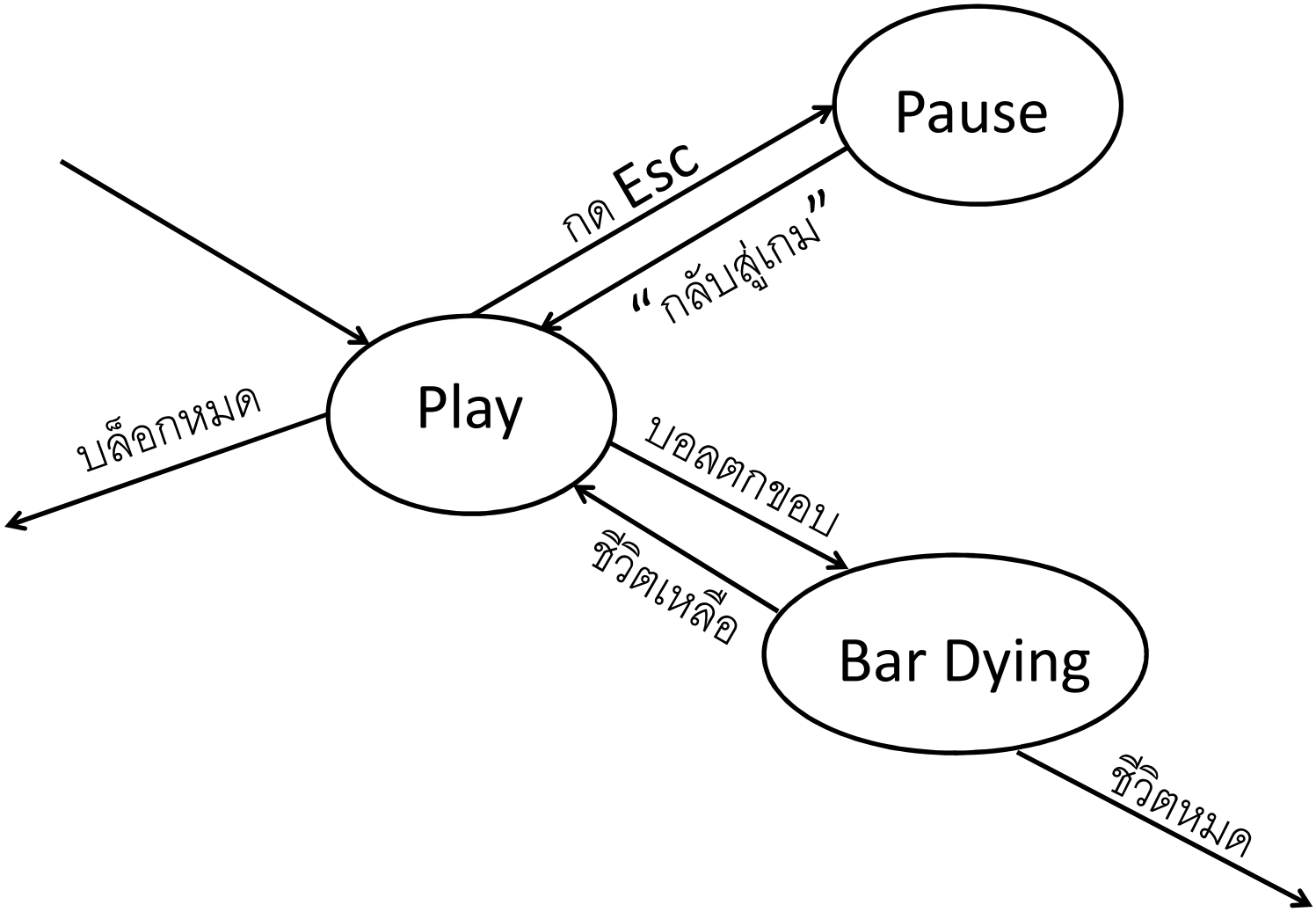
Transition Diagram ของเกม



Play Screen

- ส่วนที่ซับซ้อนที่สุดของเกม
  - มีพฤติกรรมหลายแบบ
  - คล้ายกับเกมที่มีหลายหน้าจอ
- ใช้ State Design Pattern ในการออกแบบ
- State ใน Play Screen
  - Play State = ผู้ใช้บังคับแป้นได้ เกมดำเนินไปตามกลไก
  - Bar Dying State = แสดงอนิเมชันแป้นตาย ผู้ใช้บังคับไม่ได้
  - Pause State = เกมหยุดนิ่ง มีเมนูให้ผู้ใช้เลือกว่าจะทำอะไร

Transition Diagram ของ Play Screen



การเขียนเกมที่มี “ระบบฟิสิกส์”

เกมที่มี “ระบบฟิสิกส์”

- มี “วัตถุ”
- วัตถุสามารถ “ชนกัน”
- หลังจากชนกันแล้วเกิด “เหตุการณ์”
- เหตุการณ์ทำให้สถานะของวัตถุเปลี่ยนไป

- การเคลื่อนที่ของวัตถุไม่จำเป็นต้องเป็นไปตามกฎทางฟิสิกส์จริงๆ

ตัวอย่างเกมที่มีระบบฟิสิกส์

- The Incredible Machine
- Crayon Physics

- Pong
- Breakout
- Super Mario Brothers
- First Person Shooter ต่างๆ
- เกมต่อสู้ต่างๆ

The Incredible Machine

- http://www.youtube.com/watch?v=EJbEDlDDVVc

Crayon Physics

- http://www.youtube.com/watch?v=sZaxO6wbxi8

Super Mario World's Physics System

- http://www.youtube.com/watch?v=LmR2bt3-mXY

ส่วนประกอบของเกมที่มีระบบฟิสิกส์

- วัตถุ
  - ลูกบอล ตัวละคร บล็อก พื้น แป้น ฯลฯ
  - สิ่งที่เป็นเอกเทศในตัวมันเอง สามารถมีปฏิสัมพันธ์กับสิ่งอื่นได้
  - สถานะของวัตถุคือสถานะของเกม
- กฎทางฟิสิกส์
  - บรรยายว่าวัตถุเคลื่อนที่อย่างไร
  - บรรยายว่าวัตถุต่างๆ สามารถมีปฏิสัมพันธ์กันได้อย่างไรบ้าง

ส่วนประกอบของเกมที่มีระบบฟิสิกส์

- เหตุการณ์
  - สัญญาณว่าเกิดอะไรขึ้นบ้างในระบบ
  - เป็นกลไกสำคัญของระบบฟิสิกส์ที่ร้อยส่วนต่างๆ เข้าด้วยกัน

ลูปหลักของเกมที่มีระบบฟิสิกส์

```
while (true)
{
    ให้วัตถุต่างๆ ปรับปรุงสถานะของตัวเอง (= ให้มันเคลื่อนที่)
    เช็คว่าวัตถุคู่ใดชนกันบ้าง
    จัดการกับเหตุการณ์ทั้งหมดที่เกิดขึ้น
}
```

เหตุการณ์มาจากไหน?

- เวลาวัตถุปรับสถานะของตัวเอง อาจเกิดเหตุการณ์ขึ้น
  - เวลาของไอเทมเพิ่มพลังหมด
  - เวลาของด่านทั้งด่านหมด
  - ตัวละครทำท่าทางอะไรต่างๆ เสร็จ
- การชนกันของวัตถุเป็นเหตุการณ์อย่างหนึ่ง
- เมื่อจัดการเหตุการณ์แล้วอาจทำให้เกิดเหตุการณ์อื่น
  - ตัวละครชนกับลูกปืน ทำให้ตัวละครตาย
  - ตัวละครชนกับไอเทม ทำให้พลังเพิ่ม
  - ฯลฯ

หลักการสำคัญ

- มีลิสต์ของเหตุการณ์
- เมื่อเกิดเหตุการณ์ให้เพิ่มมันเข้าในลิสต์
- ตอนท้ายลูป อ่านลิสต์นั้นแล้วจัดการมันทีละเหตุการณ์

CLASS ต่างๆ ในเกม BREAKOUT

Class ที่เกี่ยวข้อง

- public abstract class GameObject
  - บรรพบุรุษของวัตถุทั้งหมดในเกม
- public interface Event
  - บรรพบุรุษของเหตุการณ์ทั้งหมดในเกม

Class ที่เกี่ยวข้อง

- public class GameState
  - คลาสนี้เป็นที่รวม
    - GameObject ทั้งหมดในเกม
    - Event ทั้งหมดที่เกิดขึ้นในเฟรมเฟรมหนึ่ง
  - Field ที่สำคัญ
    - List<GameObject> objects;
    - List<Event> events;
    - List<GameObject> objectsToAdd;
    - List<GameObject> objectsToRemove;

objectsToAdd และ objectsToRemove

- เมื่อเกมดำเนินไป อาจมีวัตถุถูกเพิ่มเข้ามาหรือถูกลบออกไป

- แต่เราไม่ควรจะลบหรือเพิ่มวัตถุเข้าในเกมทันทีเมื่อรู้ว่าต้องทำ
  - ถ้าทำเช่นนี้การจัดการเหตุการณ์ต่างๆ อาจมีความผิดพลาด
  - ตัวอย่าง
    - ถ้าลบวัตถุออกเลย
    - อาจมีเหตุการณ์ที่เกี่ยวกับวัตถุค้างอยู่ในลิสต์ของเหตุการณ์
    - เมื่อไปจัดการเหตุการณ์นั้นกลายเป็นว่าวัตถุหายไปแล้ว

objectsToAdd และ objectsToRemove

- ดังนั้น
  - เวลาจะเพิ่มวัตถุใด เราจะเพิ่มวัตถุนั้นใส่ objectsToAdd
  - เวลาจะลบวัตถุใด เราจะเพิ่มวัตถุนั้นใส่ objectsToRemove
  - หลังจากจัดการเหตุการณ์ทุกอย่างเสร็จแล้ว
    - เราจึงเอาวัตถุใน objectsToAdd ไปใส่ใน objects
    - เราจึงลบวัตถุที่อยู่ใน objectsToRemove ทุกตัวออกจาก objects

GameObject ใน Breakout

- Bar = แป้นที่ผู้ใช้บังคับ
- Ball = ลูกบอล
- Brick = บล็อกที่ลอยอยู่ในฉาก
- Item = ไอเทมที่ผู้ใช้สามารถเก็บได้
- Border = ขอบทางด้านซ้ายขวาและด้านบนของฉาก
- DeathZone = บริเวณด้านล่างของหน้าจอที่เมื่อลูกบอลเข้าไปอยู่ในนั้นแล้วจะถือว่าออกนอกฉาก

- ทุกคลาสข้างบนเป็น subclass ของ GameObject

GameObject ใน Breakout

- สังเกต
  - เราไม่มีการเช็คว่าลูกบอลหรือไอเทมออกนอกหน้าจอหรือไม่
  - แต่เราเช็คว่ามันชนกับ DeathZone หรือไม่แทน

การปรับสถานะตนเองของวัตถุ

คลาส GameState

- private void UpdateObjects()
  - ปรับสถานะของ GameObject ทุกตัวที่อยู่ใน GameState
- private void UpdateObject(GameObject obj)
  - ปรับสถานะของ obj เพียงแต่ตัวเอง
  - การทำงานจะแตกต่างกันตามชนิดของ obj
  - กล่าวคือทำ single dispatch ตาม type ของ obj

การให้ GameObject ปรับสถานะของตนเอง

```
public GameState {
        :
        :

        void UpdateObjects() {
            foreach (GameObject obj in objects)
                UpdateObject(obj);
         }

         void UpdateObject(GameObject obj) {
            UpdateSpecificObject((dynamic)obj);
         }

        :
        :
}
```

การให้ GameObject ปรับสถานะของตนเอง

- สังเกต
  - UpdateObject จะเรียก UpdateSpecificObject โดยแปลง obj ที่ให้มาเป็น dynamic
  - หลังจากนี้เราจะต้องเขียน UpdateSpecificObject สำหรับ GameObject แบบต่างๆ ขึ้นมา

การให้ GameObject ปรับสถานะของตนเอง

```
void UpdateSpecificObject(GameObject obj) {
    // NOP
}

void UpdateSpecificObject(Ball ball) {
    ball.UpdatePosition(gameTime);
}

void UpdateSpecificObject(Item item) {
    item.UpdatePosition(gameTime);
}

void UpdateSpecificObject(Bar bar) {
    bar.Update(gameTime);
}
```

การให้ GameObject ปรับสถานะของตนเอง

- มี GameObject เพียงสามแบบที่ต้องปรับสถานะของตนเอง
  - Ball
    - ต้องเคลื่อนที่ไปตามเวลา
  - Item
    - ต้องตกลงมาข้างล่างของหน้าจอตามเวลา
  - Bar
    - เมื่อไอเทมเพิ่มพลังหมดต้องเปลี่ยนสถานเป็นแป้นปกติ

สังเกตว่าเราจะเขียน UpdateSpecificObject สำหรับ GameObject พวกนี้ไว้

การให้ GameObject ปรับสถานะของตนเอง

- สังเกตว่าเราไม่ได้เขียนโค้ดสำหรับ GameObject อื่นๆ นอกจากสามแบบข้างต้น
- นี่เป็นเพราะเราเขียน

```
void UpdateSpecificObject(GameObject obj) {
    // NOP
}
```

- เมื่อ UpdateSpecificObject ถูกเรียกกับ GameObject ประเภทอื่นๆ ที่ไม่ใช่สามแบบข้างต้น เมธอดข้างบนจะถูกเรียกแทน

การเช็คว่าวัตถุชนกันหรือไม่

GameState

- void CheckCollisions()
  - ตรวจวัตถุทุกคู่ใน objects ว่ามีตัวไหนชนกันบ้าง
  - สำหรับทุกคู่ที่ชนกัน ให้สร้าง CollisionEvent แล้วใส่ลงใน events

```
void CheckCollisions() {
    int count = objects.Count;
    for (int i = 0; i < count; i++)
      for (int j = i + 1; j < count; j++)
        if (GameObject.CheckColision(objects[i], objects[j]))
            events.Add(new CollisionEvent(objects[i], objects[j]));
}
```

CollisionEvent

```
public class CollisionEvent : Event
{
   private GameObject first;
   public GameObject First
   {
      get { return first; }
   }

   private GameObject second;
   public GameObject Second
   {
      get { return second; }
   }

   public CollisionEvent(GameObject first, GameObject second)
   {
      this.first = first;
      this.second = second;
   }
}
```

GameObject

- public static bool CheckCollision(
  GameObject first, GameObject second)
  - เช็คว่า first ชนกับ second หรือไม่
  - ถ้าชนคือ true มีเช่นนั้นคือ false
  - การทำงานขึ้นอยู่กับชนิดของ first และ second
  - กล่าวคือทำ double dispatch
  - เราใช้ dynamic เพื่อทำ double dispatch

โค้ด

```
public static bool CheckColision(
      GameObject first, GameObject second)
{
   return CheckCollisionSpecific(
      (dynamic)first, (dynamic)second);
}

private static bool CheckCollisionSpecific(
      GameObject first, GameObject second)
{
   return false;
}
```

โค้ด

- ตัวอย่าง CheckCollisionSpecific ของคู่วัตถุต่างๆ

```
private static bool CheckCollisionSpecific(
   Ball ball, Brick brick)
{
   return brick.CollisionRect.Intersects(
       ball.CollisionRect);
}
private static bool CheckCollisionSpecific(
   Brick brick, Ball ball)
{
   return CheckCollisionSpecific(ball, brick);
}
```

โค้ด

```
private static bool CheckCollisionSpecific(
   Ball ball, Border border)
{
   return !ball.StickingToBar && (
      ball.Position.X - ball.Radius < border.Left ||
      ball.Position.X + ball.Radius > border.Right ||
      ball.Position.Y - ball.Radius < border.Top);
}
private static bool CheckCollisionSpecific(
   Border border, Ball ball)
{
      return CheckCollisionSpecific(ball, border);
}
```

การจัดการกับเหตุการณ์

GameState

- void HandleEvents()
  - ดึงเหตุการณ์จาก events มาดูทีละตัว แล้วจัดการกับมันไปทีละตัว

```
void HandleEvents()
{
   int eventIndex = 0;
   while (eventIndex < events.Count)
   {
      Event ev = events[eventIndex];
      HandleEvent(ev);
      eventIndex++;
   }
}
```

- ทำไมเราไม่ใช้ for หรือ foreach แทน while?
  - ระหว่างจัดการกับเหตุการณ์อยู่อาจจะมีเหตุการณ์เกิดเพิ่มขึ้นมาก็ได้!

GameState

- void HandleEvent(Event ev)
  - จัดการกับเหตุการณ์ ev
  - เมธอดมีพฤติกรรมแตกต่างกันตาม ev
  - กล่าวคือต้องทำ single dispatch ด้วย ev
  - เราใช้ dynamic เพื่อทำ single dispatch อีกเช่นเคย

HandleEvent และ HandleSpecificEvent

```
void HandleEvent(Event ev)
{
   HandleSpecificEvent((dynamic)ev);
}

void HandleSpecificEvent(Event ev)
{
   // NOP
}

void HandleSpecificEvent(CollisionEvent ev)
{
   HandleSpecificCollision((dynamic)ev.First, (dynamic)ev.Second);
}
```

HandleSpecificEvent(CollisionEvent ev)

- การจัดการเหตุการณ์ที่วัตถุชนกันขึ้นอยู่กับชนิดของวัตถุที่ชนกัน
- กลาวคือเราต้องทำ double dispatch ตาม ev.first และ ev.second
- เราใช้ dynamic เพื่อทำ double dispatch เช่นเคย
- เราทำให้โดยปกติคู่ของของที่ชนกันไม่ทำให้เกิดอะไรขึ้นก่อน

```
void HandleSpecificCollision(
        GameObject first, GameObject second)
 {
    // NOP
}
```

โค้ด

- ตัวอย่างการจัดการการชนกันของวัตถุคู่ต่างๆ

```
void HandleSpecificCollision(Ball ball, Border border)
{
   if (ball.Position.X < border.Left + ball.Radius)
   {
      ball.Position = new Vector2(border.Left + ball.Radius, ball.Position.Y);
      ball.Velocity = new Vector2(Math.Abs(ball.Velocity.X), ball.Velocity.Y);
   }
   if (ball.Position.X > border.Right - ball.Radius)
   {
      ball.Position = new Vector2(border.Right - ball.Radius, ball.Position.Y);
      ball.Velocity = new Vector2(-Math.Abs(ball.Velocity.X), ball.Velocity.Y);
   }
   if (ball.Position.Y < border.Top + ball.Radius)
   {
      ball.Position = new Vector2(ball.Position.X, border.Top + ball.Radius);
      ball.Velocity = new Vector2(ball.Velocity.X, Math.Abs(ball.Velocity.Y));
   }
}
void HandleSpecificCollision(Border border, Ball ball)
{
   HandleSpecificCollision(ball, border);
}
```

โค้ด

```
void HandleSpecificCollision(Bar bar, Border border)
{
    if (bar.Left < border.Left)
        bar.Left = border.Left;
    if (bar.Right > border.Right)
        bar.Right = border.Right;
    foreach (Ball ball in bar.GetStickingBalls())
    {
        if (ball.Left < border.Left)
            ball.Left = border.Left;
        if (ball.Right > border.Right)
            ball.Right = border.Right;
        ball.UpdatePosition(gameTime);
    }
}
void HandleSpecificCollision(Border border, Bar bar)
{
    HandleSpecificCollision(bar, border);
}
```

เอาทุกอย่างมารวมกัน

โค้ดการ update สถานะของ GameState

```
public void Update(KeySensor keySensor,
    GameTime gameTime)
{
    objectsToAdd.Clear();
    objectsToRemove.Clear();
    events.Clear();

    this.gameTime = gameTime;
    ControlBar(keySensor);
    UpdateObjects();
    CheckCollisions();
    HandleEvents();
    RemovePendingObjects();
    AddPendingObjects();
}
```

โค้ดการ update สถานะของ GameState

```
void ControlBar(KeySensor keySensor)
{
    if (keySensor.IsKeyDown(KeyMapping.leftKey))
        bar.MoveLeft((float)ElapsedGameTimeInMilliseconds);
    if (keySensor.IsKeyDown(KeyMapping.rightKey))
        bar.MoveRight((float)ElapsedGameTimeInMilliseconds);

    if (bar.HasStickingBalls &&
    keySensor.IsKeyTyped(KeyMapping.launchKey))
    {
        Ball ball = bar.LaunchOneStickingBall();
        AddEvent(new LaunchBallEvent(ball));
    }
}
```

โค้ดการ update สถานะของ GameState

```
void AddPendingObjects()
{
    foreach (GameObject obj in objectsToAdd)
        objects.Add(obj);
}

void RemovePendingObjects()
{
    foreach (GameObject obj in objectsToRemove)
        objects.Remove(obj);
}
```