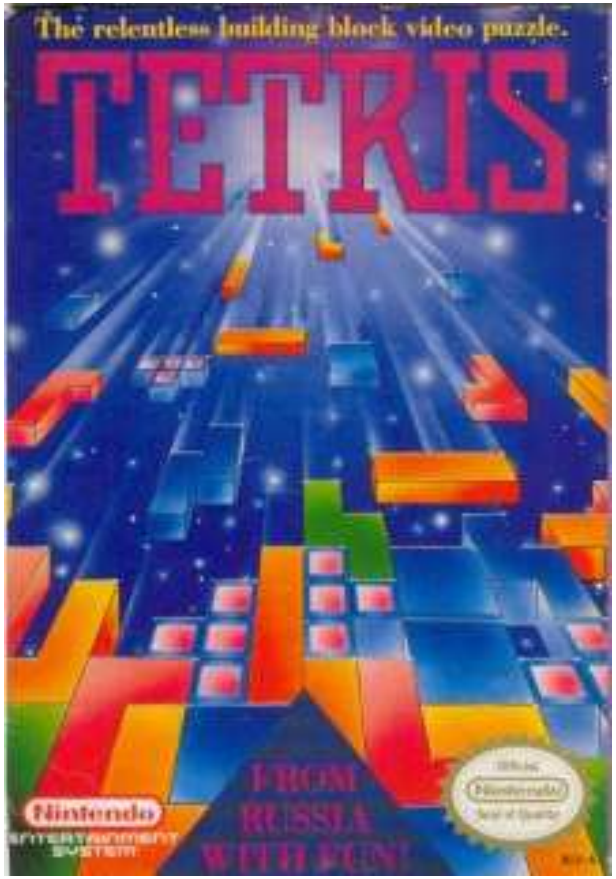
418383: การโปรแกรมเกม
การบรรยายครั้งที่ 4

ประมุข ขันเงิน

Tetris

- สร้างโดย Alexey Pajinov โปรแกรมเมอร์ชาวรัสเซีย ในปี 1985

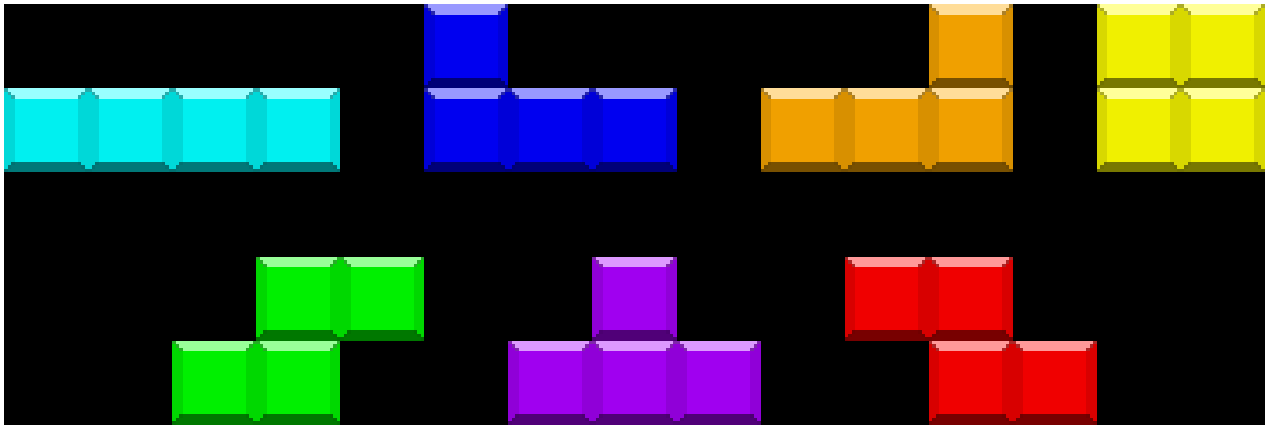


Tetris

- กฎ (จาก Wikipedia):
  - มีเทโทรมิโนตกลงมาสู่พื้นของบอร์ดทีละอัน
  - ผู้เล่นต้องควบคุมเทโทรมิโนในตกไปกองทัพกันเป็นแถวเต็มซึ่งไม่มีช่องว่างอยู่ภายใน โดยการเลื่อนไปทางซ้ายขวา หรือหมุน 90 องศา
  - เมื่อเกิดแถวเต็ม บล็อกของเทโทรมิโนในแถวนั้นทั้งหมดจะหายไป
  - เกมจะจบลงเมื่อเทโทรมิโนกองทับกันจนล้นบอร์ดบนหน้าจอ

เทโทรมิโน

- รูปทรงที่เกิดจากการเอาบล็อกสี่อันมาต่อกัน
- เรียกชื่อว่า I, J, L, O, S, T, และ Z.



Screen ต่างๆ ในเกม

- Title Screen
  - ฉากไตเติ้ล
  - คลาส TitleScreen
- Play Screen
  - ฉากเล่นเกม
  - คลาส PlayScreen
- Game Over Screen
  - ฉากเกมจบ
  - คลาส GameOverScreen

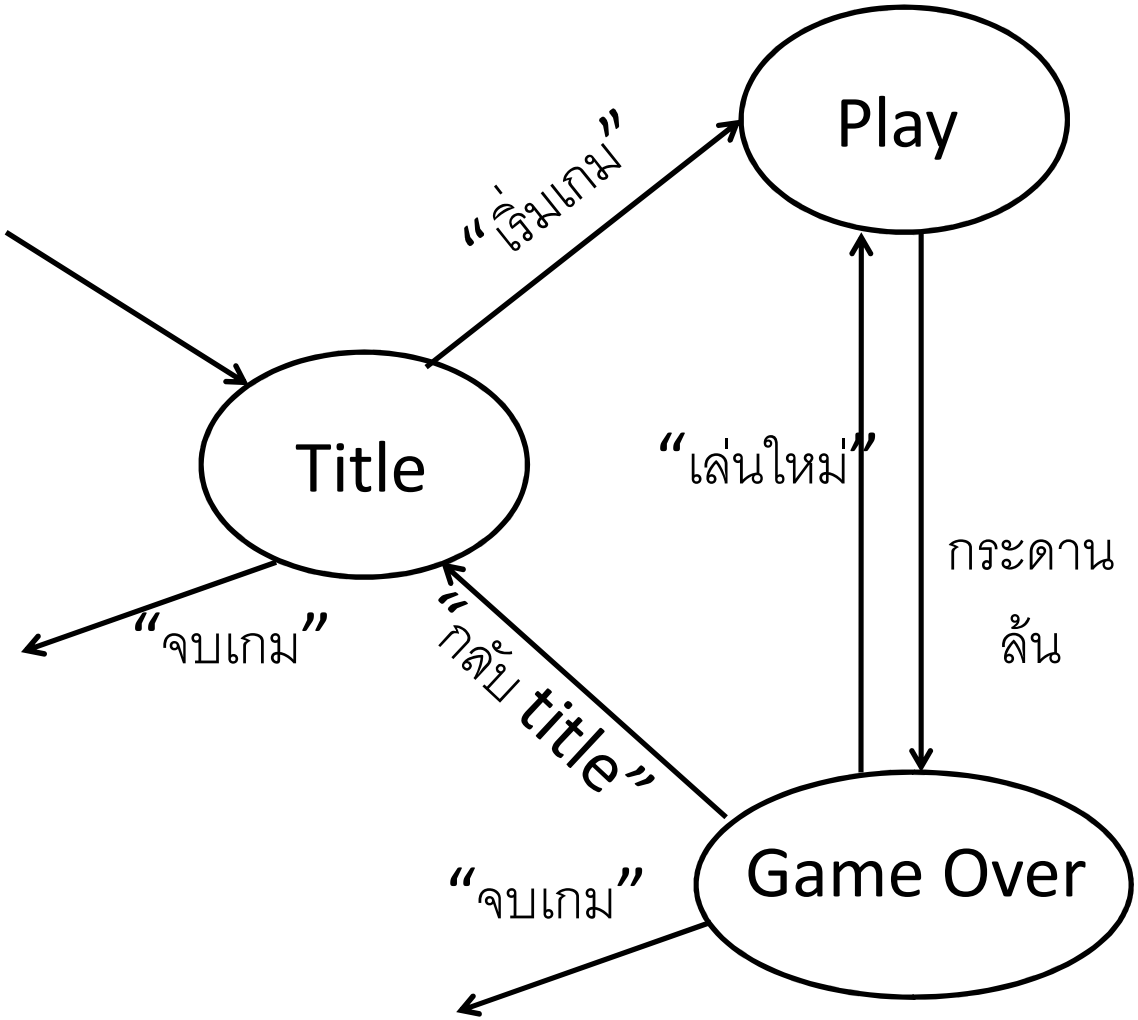
โค้ดของเกม

```
public class Tetris : GameLib.Game
{
   public Tetris()
      : base()
   {
      Content.RootDirectory = "Content";
      Graphics.PreferredBackBufferWidth = 800;
      Graphics.PreferredBackBufferHeight = 600;

      AddScreen(new Screens.TitleScreen(this));
      AddScreen(new Screens.GameOverScreen(this));
      AddScreen(new Screens.PlayScreen(this));
      SwitchScreen(“Title");

      Tetromino.InitializePrototypes();
   }
}
```

Transition Diagram ของเกม



TITLE และ GAME OVER SCREEN

เราต้องการ

มองว่าส่วนนี้เป็นเกมเกมหนึ่งเลย

- สถานะของเกม
  - ตัวเลือกที่ผู้เล่นเลือกอยู่ตอนนี้
- การจัดการปฏิสัมพันธ์กับผู้ใช้
  - ปุ่ม ขึ้น/ลง เปลี่ยนตัวเลือกที่ผู้เล่นเลือก (title)
  - ปุ่ม Enter ควรจะเปลี่ยนหน้าจอเป็นหน้าจออื่น (choices)
- การวาดภาพบนหน้าจอ
  - เขียนชื่อเกม
  - เขียนตัวเลือก และเขียนวงเล็บก้ามปูล้อมรอบตัวเลือกปัจจุบัน

จริงๆ แล้ว...

- Game Over Screen ก็มีลักษณะคล้ายๆ กัน

ระบบเมนู

- หน้าจอทั้งสองทำหน้าที่เป็นเมนู
  - อำนวยความสะดวกให้ผู้ใช้เลือกตัวเลือกจากหลายตัวเลือก
  - ตัวเลือกแต่ละตัวนำไปสู่หน้าจออื่น
- ทั้งสองหน้าจอมีหน้าตาคล้ายๆ กัน
  - ข้อความขนาดใหญ่อยู่ข้างบน
  - ข้อความแสดงตัวเลือกขนาดเล็กลงอยู่ข้างล่าง
- เราควรจะเขียนคลาสที่เก็บพฤติกรรมที่เหมือนๆ กันของระบบเมนูไว้
- แล้วจึงซับคลาสมัน เพื่อสร้างหน้าจอทั้งสอง

คลาส MenuScreen

- มีข้อมูลที่นิยามตัวเมนู และสถานะภายในของมัน
  - Title
  - Choices
  - ตัวเลือกปัจจุบัน
- จัดการข้อมูลเข้าที่ผู้ใช้ป้อน
  - เปลี่ยนตัวเลือกเมื่อผู้ใช้กดขึ้นลง
- แต่ไม่ระบุพฤติกรรมเมื่อผู้ใช้เลือกตัวเลือก
  - ตรงนี้ให้ซับคลาสไประบุเอาเอง

ฟิลด์ที่สำคัญในคลาส MenuScreen

- string title;
  - ข้อความไตเติ้ล
- string[] choices;
  - อะเรย์ของตัวเลือกต่างๆ
- int currentChoice;
  - หมายเลขของตัวเลือกปัจจุบัน

เมธอดสำคัญในคลาส MenuScreen

- public MenuScreen(GameLib.Game game,
  string name,
  string title, string titleFontName,
  string[] choices, string choiceFontName)
  - game = เกมที่มี screen นี้อยู่
  - name = ชื่อของ screen
  - title = ไตเติ้ล
  - titleFontName = asset name ของ font ที่ใช้พิมพ์ title
  - choices = ตัวเลือกต่างๆ
  - choiceFontName = assert name ของ font ที่ใช้พิมพ์ choice

เมธอดสำคัญในคลาส MenuScreen

- public abstract void ChoiceSelected(
  int index, GameTime gameTime);
  - ถูกเรียกเมื่อผู้ใช้กดปุ่ม Enter
  - index = หมายเลขของ choice ที่ถูกเลือกอยู่ในปัจจุบัน
  - เป็น abstract เพื่อให้ซับคลาสมาเติมพฤติกรรมเพิ่มเติม

Title Screen

```
public class TitleScreen : MenuScreen
   {
      public TitleScreen(GameLib.Game game)
         : base(
            game, "title_menu", "Tetris", "Vera64",
            new string[] { "Play", "Quit" }, "Vera32")
      {
      }

      :
      :
}
```

Title Screen

```
public override void ChoiceSelected(
    int index, GameTime gameTime)
{
    switch (index)
    {
        case 0:
            Game.SwitchScreen("Play");
            break;
        case 1:
            Game.Exit();
            break;
        default:
            break;
    }
}
```

Game Over Screen

```
public class GameOverScreen : MenuScreen
{
   public GameOverScreen(GameLib.Game game)
      : base(
         game, "GameOver",
         "Game Over", "Vera64",
         new string[] { "Play Again", "Return to Title", "Quit" },
         "Vera32")
   {
   }

   :
   :
}
```

Game Over Screen

```
public override void ChoiceSelected(int index, GameTime gameTime)
{
   switch (index)
   {
      case 0:
         Game.SwitchScreen("Play");
         break;
      case 1:
         Game.SwitchScreen("Title");
         break;
      case 2:
         Game.Exit();
         break;
      default:
         break;
   }
}
```

ฟอนต์

- เกมใช้ฟอนต์ Bitstream Vera
- ดาวน์โหลดได้จาก http://www.gnome.org/fonts/
- ใน TetrisContent มี Sprite Font ที่สร้างจาก Bitstream Vera อยู่สองตัว
  - Vera64
    - Bitstream Vera Sans Mono ขนาด 64p
    - ใช้เขียน title
  - Vera32
    - Bitstream Vera Sans Mono ขนาด 32p
    - ใช้เขียน choice

การจัดการกับอินพุตจาก KEYBOARD

จัดการอินพุตจากคีย์บอร์ด

- กดลูกศรขึ้น → ตัวเลือกเลื่อนขึ้น
- กดลูกศรลง → ตัวเลือกเลื่อนลง
- เราจะจัดการกับการกดปุ่มอย่างไร?

- อาจใช้ KeyboardState.IsKeyDown(Keys.Down) เพื่อตรวจว่าปุ่มลูกศรลงถูกกดหรือไม่
- ถ้าจริงก็เลื่อนตัวเลือกลง
- ทำทำนองเดียวกันได้กับปุ่มลูกศรขึ้น

- ปัญหา: ตัวเลือกถูกเปลี่ยนอยู่ตลอดเวลาเวลากดค้าง จนคนมองไม่เห็นความเปลี่ยนแปลง

จัดการอินพุตจากคีย์บอร์ด

- อีกวิธีหนึ่ง: เช็คว่าผู้ใช้เพิ่งจะกดปุ่มในเฟรมนี้หรือเปล่า
- กล่าวคือจำไว้ว่าในเฟรมก่อนกดหรือไม่
- ถ้าเฟรมก่อนไม่กด แต่เฟรมนี้กด แสดงว่าเพิ่งจะกด

- ปัญหา: ตัวเลือกไม่เปลี่ยนถ้าผู้ใช้กดปุ่มค้าง

จัดการอินพุตจากคีย์บอร์ด

- สิ่งที่เราต้องการ: พฤติกรรมเวลาเราพิมพ์ข้อความใน text editor ทั่วๆ ไป
- สมมติว่าเราใช้ Notepad จะเกิดอะไรขึ้นถ้าคุณกดปุ่ม a ค้างไว้?
  - ตัวอักษร ‘a’ ตัวแรกจะปรากฏทันทีที่กดปุ่ม
  - หลังจากนั้น ไม่มีตัวอักษร ‘a’ ปรากฏขึ้นเลยเป็นเวลาสักครึ่งวินาที
  - หลังจากนั้น ตัวอักษร ‘a’ ปรากฏขึ้นหลายตัวอย่างรวดเร็ว ด้วยความถี่สูง
  - เมื่อปล่อยปุ่ม จะไม่มีตัวอักษร ‘a’ ปรากฏขึ้นอีก

คลาส KeySensor

- โค้ดที่ใช้สร้างพฤติกรรมดังกล่าวค่อนข้างซับซ้อน
- เราจะรวมมันเป็นคลาสชื่อว่า KeySensor
- เราจะใช้ KeySensor ในการจัดการอินพุตจากคีย์บอร์ดไปตลอด

วิธีใช้ KeySensor

- ประกาศ KeySensor ไว้เป็นฟิลด์ในคลาสที่เป็น Screen
  - สมมติประกาศชื่อว่า keySensor
- ในฟังก์ชัน Update ของคลาสที่เป็น Screen ให้เรียก keySensor.Update(gameTime)
  เพื่อให้ keySensor ประมวลผลสถานะของปุ่มต่างๆ

วิธีใช้ KeySensor

- ตัวอย่าง: (MenuScreen)

```
private GameLib.KeySensor keySensor;

public override void Update(GameTime gameTime)
{
  // Update key sensor's internal information.
  keySensor.Update(gameTime);

  :
  :
}
```

วิธีใช้ KeySensor

- ลงทะเบียนให้ KeySensor “เฝ้ามอง” ปุ่มที่เราสนใจ ด้วยคำสั่ง Watch
- โดยมากจะทำใน constructor ของ Screen

วิธีใช้ KeySensor

- ตัวอย่าง: (MenuScreen)

```
public MenuScreen(...) : base(game, name)
{
   :
   :

   // Create the key sensor.
   this.keySensor = new KeySensor();
   // We will watch three keys:
   this.keySensor.Watch(Keys.Up);
   this.keySensor.Watch(Keys.Down);
   this.keySensor.Watch(Keys.Enter);

   :
   :
}
```

วิธีใช้ KeySensor

- ในฟังก์ชัน Update ของ Screen ให้ใช้ฟังก์ชันเหล่านี้ของ KeySensor ในการตรวจสถานะของปุ่ม
  - public bool IsKeyPressed(Keys key)
    - ตรวจสอบว่าคีย์ที่ให้มาเพิ่งจะถูกกดในเฟรมนั้นหรือไม่ (เฟรมก่อนไม่กด)
  - public bool IsKeyReleased(Keys key)
    - ตรวจสอบว่าคีย์ที่ให้มาเพิ่งจะถูกปล่อยในเฟรมนั้นหรือไม่ (เฟรมก่อนกดอยู่)
  - public bool IsKeyDown(Keys key)
    - ตรวจสอบว่าคีย์ที่ให้มาถูกกดอยู่ในเฟรมนั้นหรือไม่ (เช็คกดค้าง)
  - public bool IsKeyTyped(Keys key)
    - ตรวจสอบว่าคีย์ที่ให้มาถูก “พิมพ์” ในเฟรมนั้นหรือไม่
    - เมธอดนี้ทำให้เกิดพฤติกรรมเหมือนตอนพิมพ์ใน text editor

การใช้ KeySensor ใน MenuScene

```
public override void Update(GameTime gameTime)
{
    // Update key sensor's internal information.
    keySensor.Update(gameTime);

    // If the user types the up arrow,
    // move the choice upward.
    if (keySensor.IsKeyTyped(Keys.Up))
    {
        currentChoice -= 1;
        if (currentChoice < 0)
            currentChoice = choices.Length - 1;
    }
    // If the user types the down arrow,
    // move the choice downward.
    else if (keySensor.IsKeyTyped(Keys.Down))
    {
        currentChoice += 1;
        if (currentChoice >= choices.Length)
            currentChoice = 0;
    }
    // If the user pressed enter,
    // the choice is selected.
    else if (keySensor.IsKeyTyped(Keys.Enter))
        ChoiceSelected(currentChoice, gameTime);
}
```

การมาร์กเวลา

จัดการเวลา

- ในการเขียน KeySensor เราต้องสามารถ
  - มาร์กเวลาที่ปุ่มปุ่มหนึ่งถูกกดเป็นครั้งแรก
  - คำนวณว่าเวลาผ่านไปเท่าไหร่แล้วหลังจากปุ่มถูกกดครั้งแรก
- เราต้องสามารถให้ชื่อกับมาร์กเวลาที่เราทำไว้ด้วย เนื่องจาก
  - มีปุ่มหลายๆ ปุ่มที่เราต้องตรวจสอบ
  - แต่ละปุ่มจะมีมาร์กเหตุการณ์สองแบบ
    - แบบแรกสำหรับเวลาที่มันถูกกดเป็นครั้งแรก
    - แบบที่สองสำหรับเวลาที่ตัวอักษรถูก “พิมพ์” เป็นครั้งสุดท้าย

คลาส TimeMarker

- public void Mark(string eventName,
  GameTime now)
  - มาร์กเวลาปัจจุบัน (now) ด้วยชื่อที่กำหนด (eventName) ให้
- public TimeSpan GetTimeSinceLastMarked(
  string eventName, GameTime now)
  - คืนเวลาตั้งแต่เหตุการณ์ที่มีชื่อที่กำหนดให้ถูกมาร์ก
  - คืนช่วงเวลา 0 ถ้าไม่มีเหตุการณ์ที่กำหนดให้

คลาส TimeMarker

```
public class TimeMarker
{
   private Dictionary<string, double> markedTimes;

   public TimeMarker()
   {
      markedTimes = new Dictionary<string, double>();
   }

   :
   :
}
```

คลาส TimeMarker

```
public void Mark(string eventName, GameTime now)
{
    markedTimes[eventName] =
        now.TotalGameTime.TotalMilliseconds;
}

public TimeSpan GetTimeSinceLastMarked(
    string eventName, GameTime now)
{
    if (markedTimes.ContainsKey(eventName))
        return TimeSpan.FromMilliseconds(
            now.TotalGameTime.TotalMilliseconds –
                markedTimes[eventName]);
    else
        return new TimeSpan(365, 0, 0, 0);
}
```

GAME LOGIC

คลาสที่เกี่ยวกับเททริส

- Tetromino
  - แทนตัวเทโทรมิโนหนึ่งตัว
- Block
  - แทนบล็อกหนึ่งบล็อกจากตัวเทโทรมิโน
- TetrisBoard
  - แทนบอร์ดที่ใช้เล่นเททริส

คลาส Block

- แทนบล็อกหนึ่งบล็อกที่มาจากตัว Tetromino
- ฟิลด์
  - x = ตำแหน่งตามแกน X
  - y = ตำแหน่งตามแกน Y
  - shape = ชนิดของเทโทรมิโนต้นกำหนด (I, J, L, O, S, T, หรือ Z)

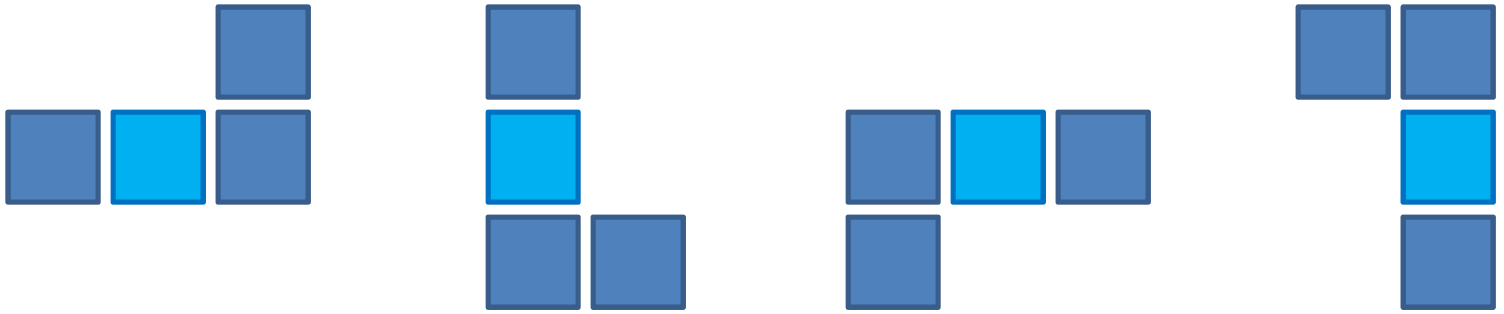
TetrominoShape

- ชนิดข้อมูลแบบ enum ที่เก็บชนิดของตัวเทโทรมิโนไว้ทั้งหมด

```
public enum TetrominoShape
{
    I,
    J,
    L,
    O,
    S,
    T,
    Z
}
```

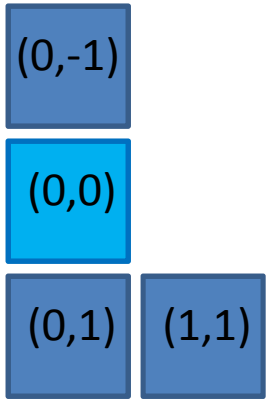
เข้ารหัสเทโทรมิโน

- เทโทรมิโนแต่ละตัวมี “จุดหมุน” ซึ่งเป็นบล็อกที่เวลาหมุนเทโทรมิโนแล้วบล็อกอื่นจะหมุนรอบบล็อกนั้น.
- ยกตัวอย่างเช่น เทโทรมิโน **L** จะมีบล็อกดังเห็นข้างล่างนี้เป็นจุดหมุน



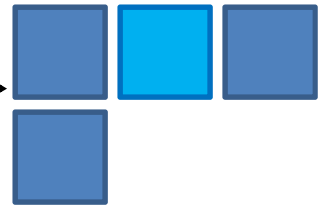
เข้ารหัสเทโทรมิโน

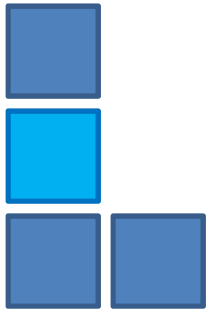
- สำหรับเทโทรมิโนแต่ละตัว เราจะสร้างระบบพิกัดของมัน
- ให้จุดหมุนมีพิกัด (0,0).
- พิกัดของบล็อกอื่นๆ คิดเทียบตามบล็อกนั้น
- ตัวอย่าง

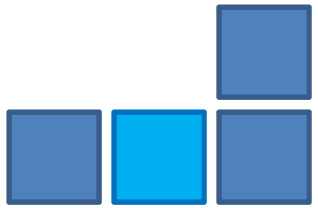


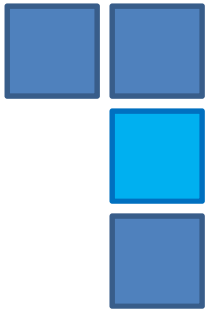
เข้ารหัสเทโทรมิโน

- เทโทรมิโน **L** ถูกหมุนได้ **4** แบบ ดังนั้นสามารถแทนได้ด้วยบล็อกต่างๆ ดังจะเห็นได้ข้างล่างนี้

{(0,0),(-1,0),(1,0),(-1,1)} 

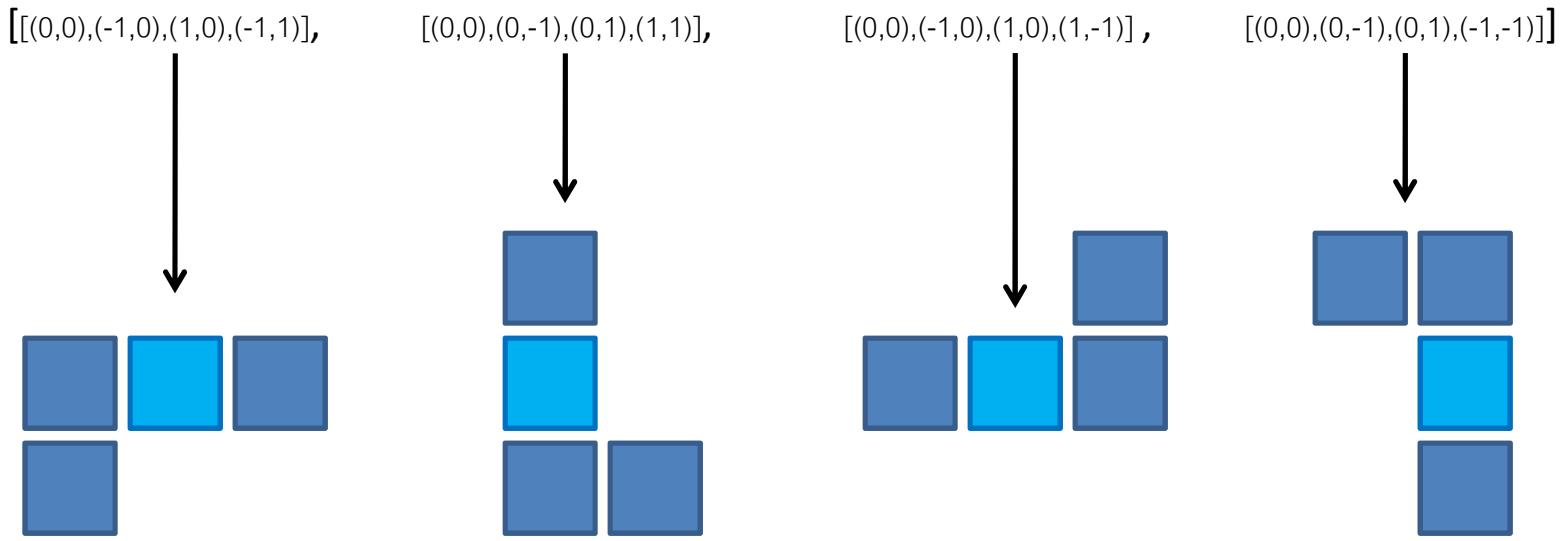
{(0,0),(0,-1),(0,1),(1,1)} 

{(0,0),(-1,0),(1,0),(1,-1)} 

{(0,0),(0,-1),(0,1),(-1,-1)} 

เข้ารหัสเทโทรมิโน

- เพื่อทำให้การหมุนเทโทรมิโนง่าย เราจะเก็บลิสต์ของบล็อกที่หมุนแล้วในลิสต์ซึ่งเรียงตามการหมุนทวนเข็มนาฬิกา



เข้ารหัสเทโทรมิโน

- เราทำเช่นนี้กับเทโทรมิโนทุกแบบ แล้วเก็บข้อมูลไว้ในดิกชันนารีชื่อ prototypes ซึ่งเป็น static field ของคลาส Tetromino

```
private static Dictionary<TetrominoShape,
    List<List<Block>>> prototypes = null;
```

- การสร้างดิกชันนารีนี้และการเติมมันให้เต็ม จะถูกทำใน static method ชื่อ InitializePrototypes ซึ่งถูกเรียกใน constructor ของเกม

คลาส Tetromino

- แทนตัวเทโทรมิโนหนึ่งตัว
- มีฟิลด์สี่ฟิลด์
  - shape = รูปร่างของเทโทรมิโน (ชนิด TetrominoShape)
  - rotation = จำนวนเต็มที่บอกว่าตอนนี้เทโทรมิโนอยู่ในการหมุนที่เท่าไหร่ โดยหมายเลขการหมุนนี้อ้างจากตำแหน่งของการหมุนใน prototypes
  - x = ตำแหน่งตามแกน X ของจุดหมุนในบอร์ด
  - y = ตำแหน่งตามแกน Y ของจุดหมุนในบอร์ด

คลาส Tetromino

- public void RotateClockwise()
- public void RotateCounterClockwise()
  - หมุนบล็อกตามเข็มและทวนเข็มนาฬิกา
  - ทำโดยการเพิ่มหรือลดค่า rotation ทีละ 1

- public IEnumerable<Block> GetBlocks()
  - คืนบล็อกทั้งหมดในเทโทรมิโนมา
  - บล็อกที่คืนมาจะมีตำแหน่ง xy อยู่ในระบบพิกัดของบอร์ด

คลาส TetrisBoard

- แทนบอร์ด (กระดาน) ที่เราใช้เล่นเททริส
- มีฟิลด์สามฟิลด์
  - blocks = ลิสต์ของบล็อกในบอร์ด**ที่ตกถึงพื้นแล้ว** (ไม่รวมที่ผู้เล่นบังคับ)
  - width = ความกว้าง
  - height = ความสูง

การตรวจการชนกันของเทโทรมิโนกับบอร์ด

- public bool CheckSideCollision(
  Tetromino tetromino)
  - เช็คว่าเทโทรมิโนที่ให้ชนกับขอบด้านข้างของบอร์ดหรือไม่
  - ชน = มีบล็อกหนึ่งบล็อกเลยขอบด้านข้างไป
- public bool CheckBottomCollision(
  Tetromino tetromino)
  - เช็คว่าเทโทรมิโนที่ให้ชนกับขอบด้านล่างบอร์ดหรือไม่
  - ชน = มีบล็อกหนึ่งบล็อกเลยขอบด้านล่างไป

การตรวจการชนกันของเทโทรมิโนกับบอร์ด

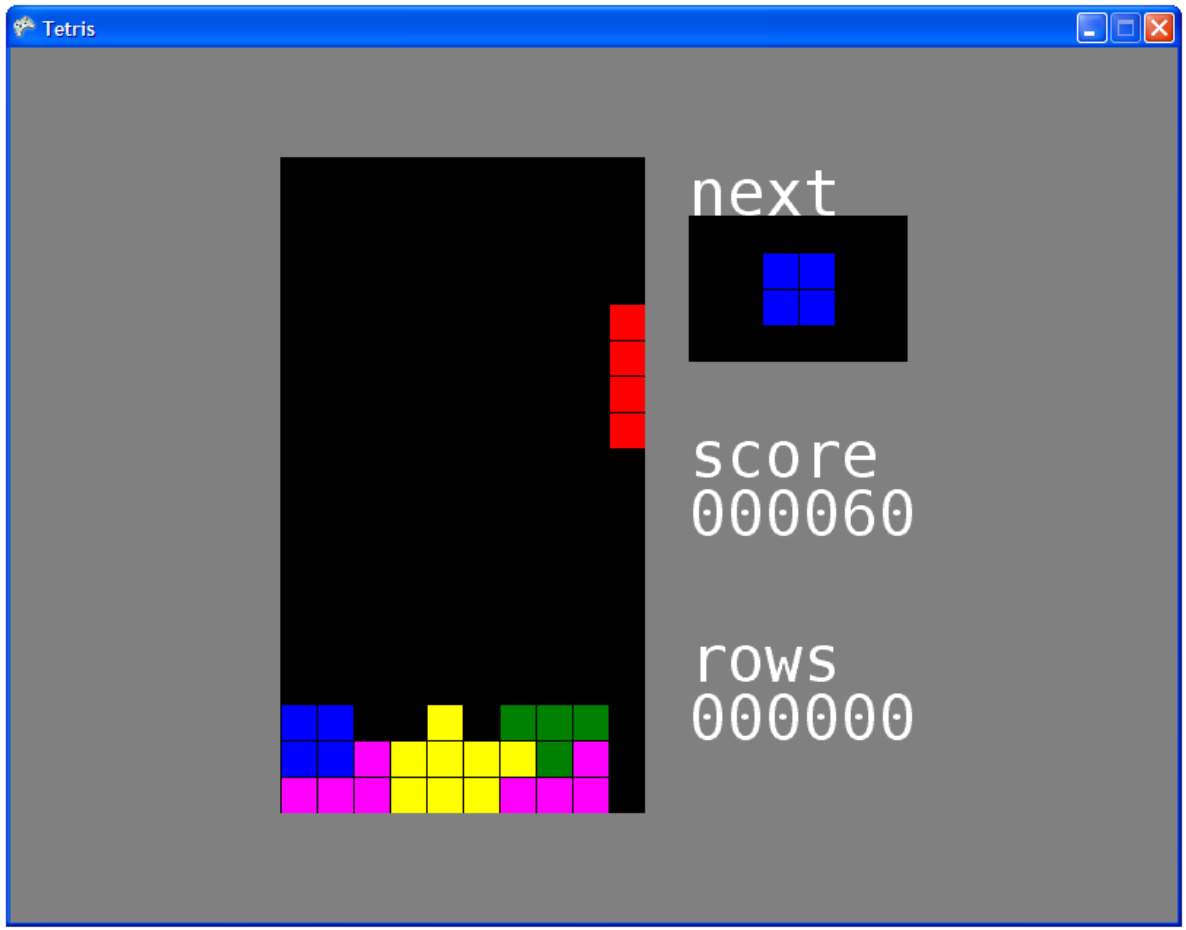
- public bool CheckBlockCollision(
  Tetromino tetromino)
  - เช็คว่าเทโทรมิโนที่ให้ชนกับบล็อกใดบล็อกหนึ่งที่อยู่ในบอร์ดแล้วหรือไม่
  - ชน = มีบล็อกของเทโทรมิโนซ้อนทับกับบล็อกที่มีอยู่ในบอร์ดแล้วพอดี
- public bool CheckCollision(
  Tetromino tetromino)
  - เช็คว่าเทโทรมิโนที่ให้ชนกับอะไรในบอร์ดหรือไม่
  - ทำการเช็คสามอย่างที่แล้วทั้งหมด

เมธอดสำคัญอื่นๆ

- public void Freeze(Tetromino tetromino)
  - ย่อยเทโทรมิโนที่กำหนดให้เป็นบล็อก แล้วนำบล็อกไปใส่ในบอร์ด
  - ใช้เวลาเทโทรมิโนตกถึงพื้น
- public List<int> GetFullRows()
  - คืนลิสต์ของหมายเลขของแถวที่เต็มแล้ว
- public void RemoveFullRows()
  - ลบบล็อกในแถวที่เต็มแล้วออกจากบอร์ด

PLAY SCREEN

Play Screen



Play Screen

- ฟิลด์ที่สำคัญ
  - board = บอรด์ (instance ของ TetrisBoard)
  - playerPiece = เทโทรมิโนที่ผู้ใช้ควบคุม
  - nextPiece = เทโทรมิโนอันต่อไป (แสดงอยู่ด้านข้าง)
  - descendDelay = เวลาจนกว่าเทโทรมิโนจะเลื่อนลงข้างล่างเอง (1 วิ)
  - fullRowsCreated = แถวเต็มที่ทำได้แล้ว
  - score = คะแนนที่ทำได้

“โหมด” ของ Play Screen

- การควบคุมของผู้ใช้ใน Play Screen แบ่งออกได้เป็น 2 โหมดใหญ่
  - Play Mode = ผู้เล่นเล่นธรรมดา
  - Full Row Mode = ผู้เล่นทำอะไรไม่ได้เลยระหว่างที่เกมแสดงว่ามีแถวเต็มและมันกำลังจะหายไป
- ฟิลด์ที่ช่วยจัดการสองโหมดนี้
  - mode = โหมดปัจจุบัน จะมีค่าเป็น PlayScreenMode.Play หรือ PlayScreenMode.FullRow
  - blinkDelay = เวลาในการกระพริบครึ่งรอบของแถวเต็ม
  - blinkCounter = แถวเต็มกระพริบไปกี่ครึ่งรอบแล้ว

เมธอดที่สำคัญ

- public void ResetGame()
  - เริ่มเกมใหม่
  - เคลียร์บอร์ด คะแนน ฯลฯ
- private void PrepareNextPiece()
  - เอา nextPiece ไปใส่ playerPiece
  - สร้างเทโทรมิโนอันใหม่แล้วเอาไปใส่ nextPiece

Update

- การเปลี่ยนแปลงสถานภายในของเกมขึ้นอยู่กับโหมด
  - Play Mode
    - รับอินพุตจากคีย์บอร์ด
    - ควบคุมเทโทรมิโน
    - เช็คแถวเต็มและเกมโอเวอร์
  - Full Row Mode
    - รอให้เวลาผ่านไปเฉยๆ
    - ถ้าเวลาผ่านไปเกิน blinkDelay ให้เพิ่ม blinkCounter

Play Mode

- การ update สถานะขึ้นอยู่กับการเคลื่อนที่ของเทโทรมิโน
  - หมุน
  - ไปทางซ้าย
  - ไปทางขวา
  - เลื่อนลง

หมุนเทโทรมิโน

- เช็คปุ่มลูกศรขึ้น
- ถ้าหมุนแล้วไม่ไปชนอะไรก็หมุนได้

- ใน Update

```
if (keySensor.IsKeyTyped(Keys.Up))
    RotateIfOkay();
```

- เมธอด RotateIfOkay

```
private void RotateIfOkay()
{
    playerPiece.RotateCounterClockwise();
    if (board.CheckCollision(playerPiece))
        playerPiece.RotateClockwise();
}
```

เลื่อนไปทางซ้ายหรือขวา

```
// Move left if the user types the left arrows.
if (keySensor.IsKeyTyped(Keys.Left))
{
    playerPiece.X -= 1;
    if (board.CheckCollision(playerPiece))
        playerPiece.X += 1;
}
// Move right if the user types the right arrows.
else if (keySensor.IsKeyTyped(Keys.Right))
{
    playerPiece.X += 1;
    if (board.CheckCollision(playerPiece))
        playerPiece.X -= 1;
}
```

เลื่อนลง

- มีสองกรณี
  - กรณีที่ผู้ใช้ “พิมพ์” ลูกศรลง
  - กรณีที่เทโทรมิโนมันเลื่อนลงมาเองเมื่อเวลาผ่านไป descendDelay
- กรณีแรกเช็คง่าย
- กรณีที่สองต้องใช้ TimeMarker ช่วย
- ใช้ TimeMarker จำเวลาครั้งที่แล้วที่เทโทรมิโนเลื่อนลง
- ถ้าเวลาผ่านจากนั้นไปเกิน descendDelay ให้เลื่อนลง
- เลื่อนลงให้มาร์กเวลาเลื่อนลงใหม่ทันที

เลื่อนลง

```
if (keySensor.IsKeyTyped(Keys.Down) ||
    timeMarker.GetTimeSinceLastMarked(
        LastDescentEventName, gameTime) > descendDelay)
{
    timeMarker.Mark(LastDescentEventName, gameTime);
    playerPiece.Y += 1;

    :
    :
}
```

เลื่อนลง

- เมื่อเทโทรมิโนเลื่อนลงแล้วต้องเช็ค
  - มันชนกับอะไรหรือไม่?
  - ถ้าชน
    - ต้อง Freeze มัน
    - เช็คว่ามีแถวเต็มหรือไม่
      - ถ้าใช่ให้เปลี่ยนเป็น Full Row Mode
    - เช็คว่า Game Over หรือไม่
      - ถ้าใช่ให้เปลี่ยนเป็น Game Over Screen

เลื่อนลง

```
if (board.CheckCollision(playerPiece))
{
    playerPiece.Y -= 1;
    board.Freeze(playerPiece);
    PrepareNextPiece();
    score += PieceScore();

    var fullRows = board.GetFullRows();
    if (fullRows.Count > 0)
    {
        mode = PlayScreenMode.FullRow;
        ResetFullRowMode(gameTime);
    }

    if (board.CheckCollision(playerPiece))
        Game.SwitchScreen("GameOver");
}
```

Full Row Mode

```
if (timeMarker.GetTimeSinceLastMarked(
      LastBlinkEventName, gameTime) > blinkDelay)
{
   timeMarker.Mark(LastBlinkEventName, gameTime);
   blinkCounter++;
}

if (blinkCounter >= 4)
{
   int fullRowCount = board.GetFullRows().Count;
   fullRowsCreated += fullRowCount;
   score += FullRowScore(fullRowCount);
   board.RemoveFullRows();
   mode = PlayScreenMode.Play;
   timeMarker.Mark(LastDescentEventName, gameTime);
}
```