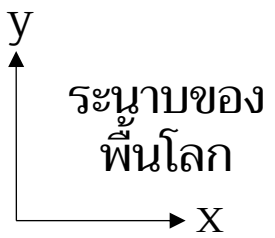
10. Tilemap

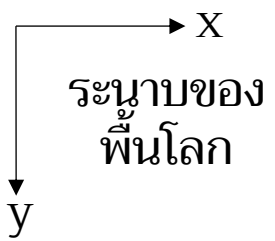
Game Camera View
มุมมองกล้องในเกม

1. Top View

ระนาบพื้นโลกเป็นระนาบเดียวกับจอภาพ

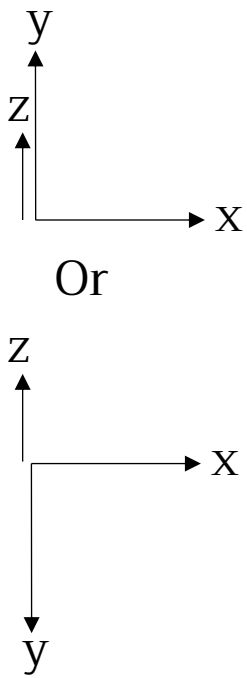


Or



XY ในที่นี้ไม่ใช่ XY ของ SFML
แต่กำหนดให้ XY แทนระนาบของพื้นโลก
ส่วนแกน Z ถ้ามี จะหมายถึงแกนตั้งฉากกับพื้นโลก
ชื่อแกนจะเปลี่ยนชื่ออื่นก็ได้ ไม่ตายตัว

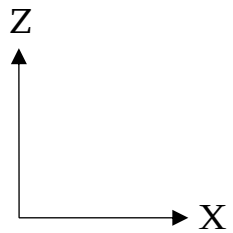
Top View with Side



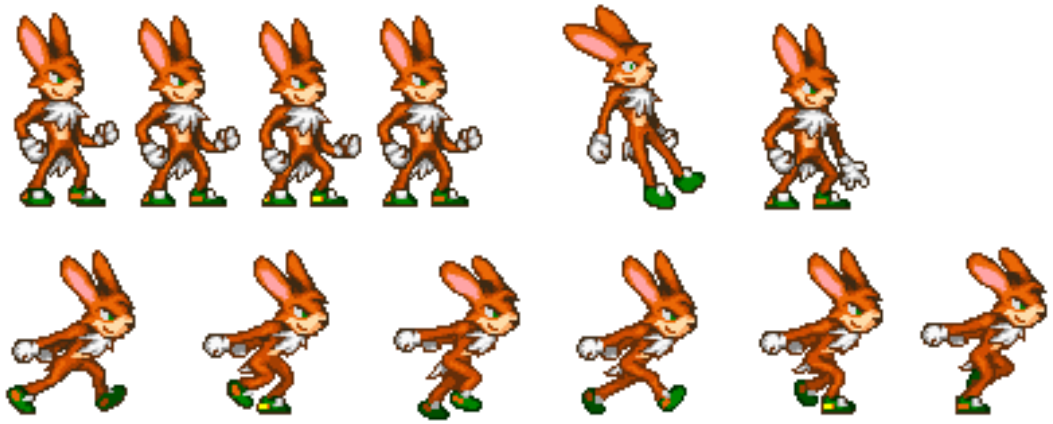
ข้อดีคือ

- มองเห็นความสูงของตัวละครได้ด้วย
- เขียนโค้ดการเคลื่อนที่ได้ไม่ยาก เพราะ X-Y ตรงกับ X-Y ของจอภาพ

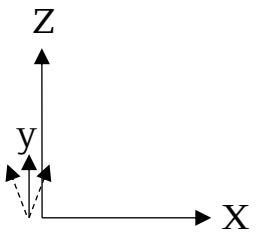
2. Side View




SPEED
x1
x2
CLICK BOB TO GET
HIM GOING AGAIN
LEVEL
2
EXIT>
SCORE: 257
A10.com
WALKTHROUGH

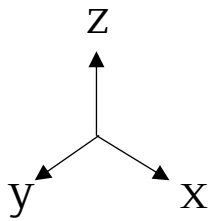


Side View with Depth




Fight
Escape
Harold HP 450 MP 90 TP 23
Therese HP 450 MP 90 TP 8
Marsha HP 450 MP 90 TP 22
Lucius HP 450 MP 90 TP 14

3. Isometric View

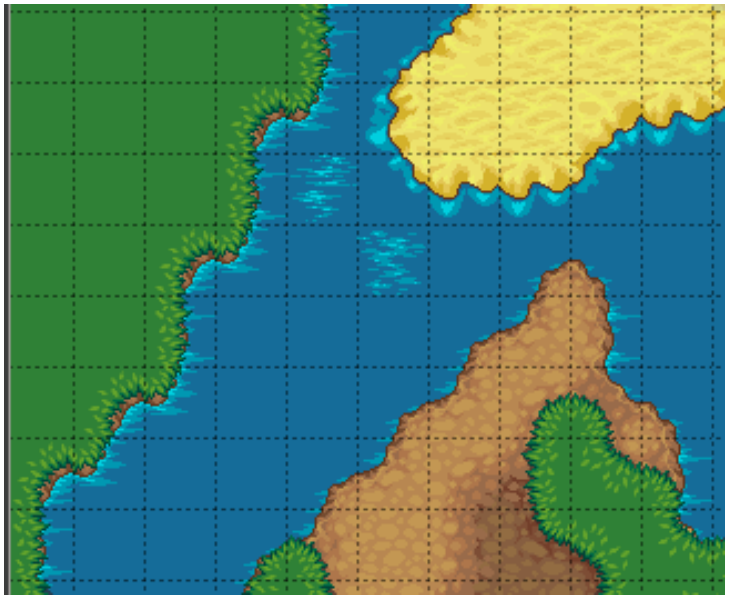



OUT OF ORDER
60
100

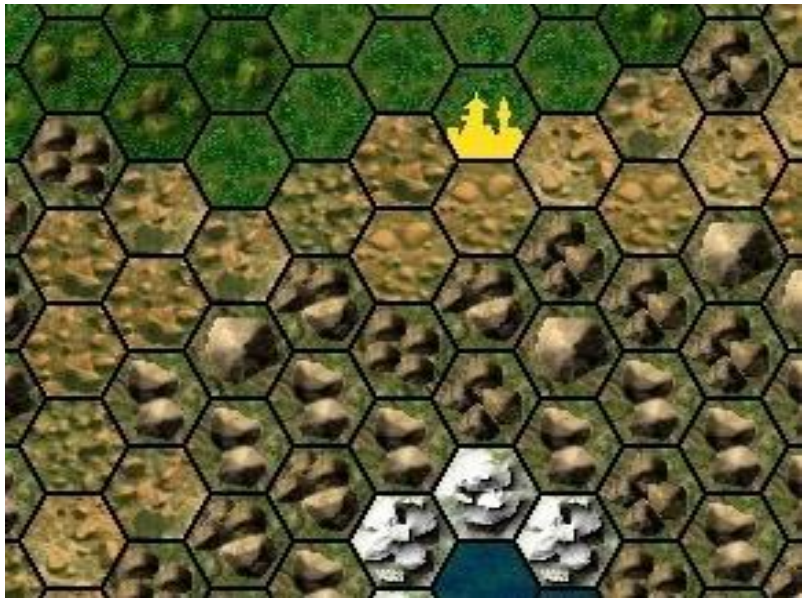
Tilemap (หรือ Tile Map)

ภาพใหญ่ที่เกิดจากการใช้ภาพเล็กๆ ซ้ำๆ กัน มีหลายประเภท เช่น

1. Orthogonal (rectangle)
2. Hexagonal
3. Isometric



Orthogonal(square) Tilemap



Hexagonal Tilemap

http://timjonsson.deviantart.com/art/Isometric-game-old-413395235

Isometric Tilemap

Tilemap vs. Tileset

- Tilemap คือภาพใหญ่ที่ประกอบด้วย tile (รูปเล็กๆ)
- Tileset คือ set ของ tile ทั้งหมดที่มีให้เลือกใช้งาน
    - Tile แต่ละรูปแบบ ถูกกำหนดอยู่ใน TileSet

Tilemap

Tileset

Source: https://opengameart.org/content/just-some-32x32-tiles

ประโยชน์และข้อจำกัดของ Tilemap

ประโยชน์

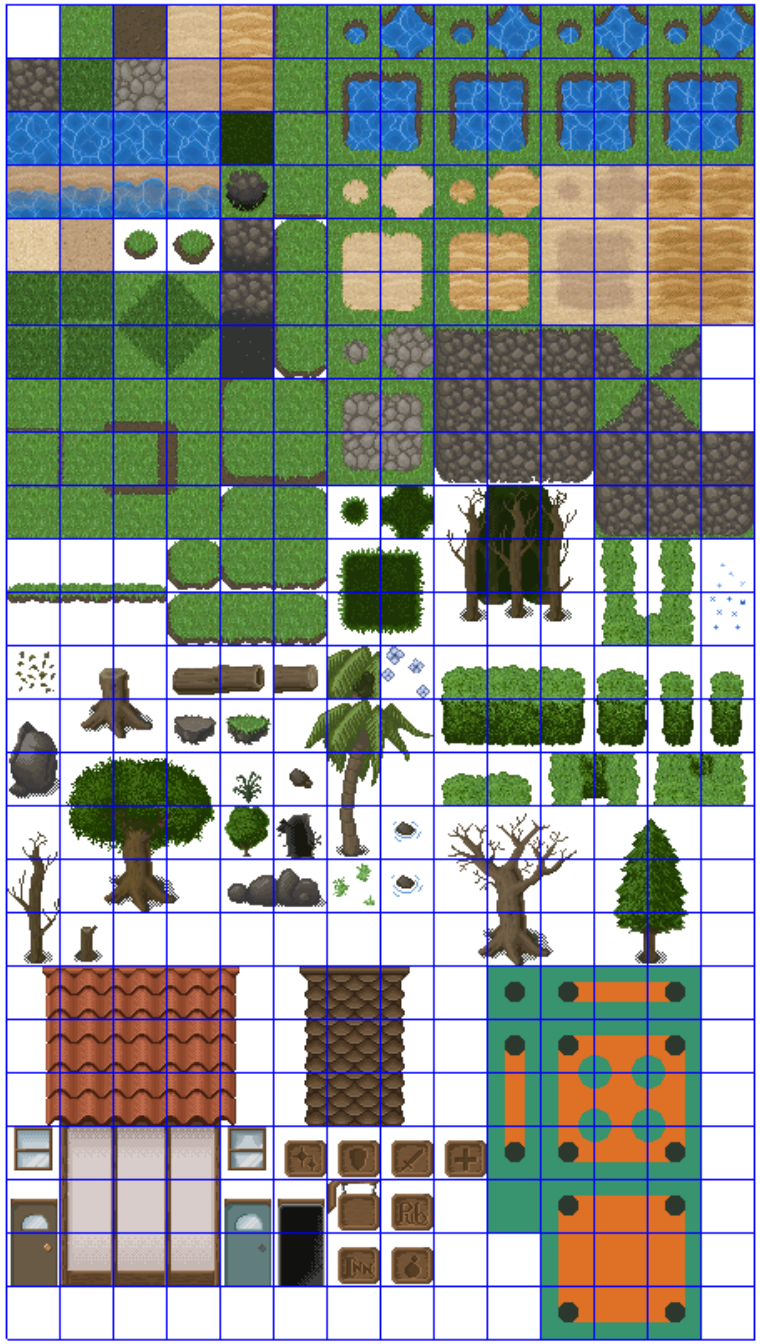
- ใช้สร้างฉากได้หลากหลายรูปแบบ ด้วยภาพเล็กๆ จำนวนไม่มาก ทำให้ไม่เสียเวลาในการวาดภาพมาก
- ตรวจเช็ค collision อย่างง่ายๆ ได้ (โดยเขียน code เพิ่ม)

ข้อจำกัด

- ภาพจะต้องวางในตำแหน่งพอดีแต่ละช่อง ไม่สามารถแบ่งหนึ่งช่องเป็นสองภาพเล็ก ภาพละครึ่งช่องได้

รูปแบบการใช้งาน Tilemap

1. Visual Only
    - แสดงผลอย่างเดียว ไม่สนใจเรื่องเช็คการชน
2. Collision Detection only
    - ไม่แสดงผล (เป็น invisible tilemap)
    - ค่าตาราง tilemap มีไว้เพื่อเช็คการชน เท่านั้น
3. Visual & Collision Detection
    - แสดงผลภาพด้วย
    - ตรวจสอบการชนกันได้ด้วย จากค่าในตาราง tilemap



Sample Tile Set

- 14 * 25 ช่อง
- ช่องละ 32*32 pixels

Source: https://opengameart.org/content/just-some-32x32-tiles

คลาส TileMap

คลาส TileMap ทำหน้าที่แสดงผล Tile ต่างๆ ในรูปตาราง

- tileSize เป็น Vector2 บอกขนาด tile 1 ช่อง
- TileMap จะรับค่า int array 2 มิติ เรียกว่า tile array ซึ่งภายในระบุรหัส tile code
- TileMap รับ function ที่ทำหน้าที่สร้าง Tile จากรหัส tile code
    - Tile ในที่นี้จะเป็น Actor แบบใดก็ได้

```
var tileArray = new int[3, 4] {
    {2,2,3,2},
    {3,1,1,3},
    {2,2,1,2}
};
tileMap = new TileMap(tileSize, tileArray, CreateTile);
```

Prototype ของ Tilemap constructor เป็นดังนี้

```
public TileMap(Vector2 tileSize, int[,] tileArray, CreateTileDelegate createTile)
```

Coding: TileMap

```
public class Game10 : Game2D
{
    TextureRegion[] tiles;
    TileMap tileMap;
    Player player;
    0 references
    protected override void LoadContent()
    {
        var tileSize = new Vector2(128, 128);
        BackgroundColor = Color.White;
        player = new Player() { Position = tileSize / 2 };

        PrepareTileSet();
        var tileArray = new int[3, 4] {
            {2,2,3,2},
            {3,1,1,3},
            {2,2,1,2}
        };
        tileMap = new TileMap(tileSize, tileArray, CreateTile);

        var visual = new Actor() { Position = new Vector2(200, 200) };
        visual.Add(tileMap);
        visual.Add(player);
        All.Add(visual);
    }
```

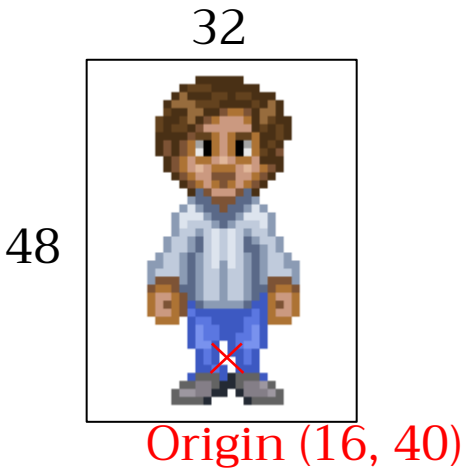
```
private void PrepareTileSet()
{
    var texture = TextureCache.Get("TileSet.png");
    var tiles2d = RegionCutter.Cut(texture, new Vector2(32, 32), countX: 14, countY: 25);
    tiles = RegionSelector.SelectAll(tiles2d);
}
1 reference
private Actor CreateTile(int tileCode)
{
    var sprite = new SpriteActor(tiles[tileCode]);
    sprite.Origin = sprite.RawSize / 2;
    sprite.Scale = new Vector2(4, 4);
    return sprite;
}
```

เราจะตั้ง origin ที่กึ่งกลางรูปภาพ เพื่อเป็นจุดอ้างอิงกับ Player

คลาส Player

สร้างคลาส Player ให้เป็น SpriteActor

- ตั้ง Origin โดยประมาณให้อยู่กึ่งกลางช่อง เวลาคนยืนบนช่อง (tile)




```
public class Player : SpriteActor
{
    1 reference
    public Player()
    {
        var texture = TextureCache.Get("Guy.png");
        SetTextureRegion(new TextureRegion(texture, new RectF(0, 0, 32, 48)));
        Origin = new Vector2(16, 40);
        Scale = new Vector2(4, 4);
    }
}
```

กำหนดค่า Origin เพื่ออะไร?

กำหนดค่า Origin เพื่อให้ตำแหน่งของ Player กับตำแหน่งของ Tile เมื่อ Position เท่ากันแล้วดูแล้วเหมือน Player ยืนบน Tile นั้นพอดี

- เนื่องจาก Tile แต่ละรูป เรากำหนด Origin ที่ศูนย์กลางของสี่เหลี่ยม
- ภาพ Player เนื่องจากมีขนาดใหญ่กว่าสี่เหลี่ยม Tile จึงต้องลองปรับค่า Origin ให้พอเหมาะ เพื่อให้ดูเหมือนผู้เล่นยืนบน Tile นั้นๆ

Grid-based Movements

การเคลื่อนที่ที่มีจุด check point อยู่บนตารางระยะห่างเท่าๆ กัน (grid)

แบ่งเป็นรูปแบบย่อยได้หลายแบบ

1. แบบ Step Jump : กดลูกศรแล้วตำแหน่งเปลี่ยนไปช่องถัดไปทันที
2. แบบ Smooth : คือ กดลูกศรแล้ว ค่อยๆ วิ่งไปช่องถัดไป
    a) เมื่อถึงช่องถัดไปแล้วจะหยุด ต้องยกคีย์แล้วกดซ้ำจึงจะเคลื่อนไปช่องถัดไป
    b) เมื่อถึงช่องถัดไปแล้วจะหยุด แต่หากกดคีย์ค้างไว้ จะเคลื่อนต่อไปได้เรื่อยๆ
    c) วิ่งต่อเนื่องไปเรื่อยๆ (แม้ไม่กดคีย์ค้าง) จนชนอะไรบางอย่าง

1. Movement แบบ Step Jump

กดปุ่มหนึ่งครั้ง ก็กระโดดไปช่องถัดไป ตามทิศทางปุ่มกดเลย

- Coding ง่าย เพราะ mapping 1-1 ระหว่างเหตุการณ์ KeyPressed() กับพฤติกรรมการเปลี่ยนตำแหน่งของตัวละคร

```
private void StepJumpMovement()
{
    var keyInfo = GlobalKeyboardInfo.Value;
    if (!keyInfo.IsAnyKeyPressed())
        return;

    var key = keyInfo.GetPressedKeys()[0];
    var direction = DirectionKey.DirectionOf(key);
    if(!IsAllowMove(direction))
        return;
    player.Position += direction * tileMap.TileSize;
}
```

ส่วนนี้เป็นการตรวจสอบการชน ค่อยทำหลังจากทำโค้ดหน้าถัดไปแล้ว

ปรับตำแหน่งขยับไป 1 ช่อง ในทันที

การตรวจสอบการออกนอกบริเวณ และการตรวจสอบการชน
(Grid-based collision detection)

การเคลื่อนที่ที่ไม่อนุญาต มีสองลักษณะ

1. ออกนอกขอบเขตของ TileMap
2. ชนกับ Tile ที่ไม่อนุญาตให้เดินเข้าไป

การตรวจสอบการชน และการออกนอกขอบเขต (ใช้ร่วมกับโค้ดหน้าที่แล้ว)

```
private bool IsAllowMove(Vector2 direction)
{
    Vector2i index = tileMap.CalcIndex(player.Position, direction);
    return tileMap.IsInside(index) && IsAllowTile(index);
}

1 reference
private bool IsAllowTile(Vector2i index)
{
    int tileCode = tileMap.GetTileCode(index);
    return tileCode != 2;
}
```

1

2

2. Movement แบบ Smooth

- เมื่อกดปุ่มลูกศร จะค่อยๆ เลื่อนตำแหน่งไปจนถึงอีกช่องหนึ่ง

- มักมีการใช้ queue เพื่อรับ key ล่วงหน้า ก่อนถึงทางเลี้ยว

ที่ Update():

1. การใช้ KeyQueue เพื่อเก็บค่าปุ่มกดล่วงหน้า เช่นกดปุ่มเลี้ยว ก่อนจะถึงจุดที่เลี้ยวได้
2. การเรียก motion.Act() เพื่อให้มีการขยับ player หากยังเคลื่อนที่ไม่เสร็จสิ้น

```
KeyQueue keyQueue = new KeyQueue();
LinearMotion motion = LinearMotion.Empty();

3 references
protected override void Update(float deltaTime)
{
    (1) var keyInfo = GlobalKeyboardInfo.Value;
        keyQueue.EnqueueAll(keyInfo.GetPressedKeys());

    (2) motion.Act(deltaTime);
        SmoothMovement();

        //StepJumpMovement();
}
```

```
private void SmoothMovement()
{
    if (!motion.IsFinished()) // ถ้าไม่มีบรรทัดนี้ จะกดคีย์ใหม่ก่อนเคลื่อนที่เสร็จได้
    {
        var command1 = keyQueue.PeekCommand();
        if (command1.IsOpposite(motion.Direction))
            UnstableMoveOpposite(keyQueue.GetCommand().Direction);
        return;
    }

    var command = keyQueue.GetCommand();
    Vector2 direction = Vector2.Zero;
    if (command.HasCommand())
        direction = command.Direction;
    else
    {
        direction = DirectionKey.Direction4;
        if (DirectionKey.Direction4 == Vector2.Zero)
            direction = motion.Direction;
    }

    StableMove(direction);
}
```

กรณีต้องการเคลื่อนที่กลับทิศได้ในทันที

2.a ถ้าละส่วน else ทั้งหมด จะทำให้ต้องกด key ซ้ำๆ ทุกครั้งที่จะขยับหนึ่งช่อง

2.b บรรทัดนี้ ทำให้กด key ค้างได้

2.c if ส่วนนี้ ทำให้แม้ไม่กด key ค้างแต่ player จะเคลื่อนที่ไปจนสุดได้

```
private void StableMove(Vector2 direction)
{
    if (!IsAllowMove(direction))
        direction = new Vector2(0, 0);

    if (motion.Direction == Vector2.Zero && direction == Vector2.Zero)
        return; // ถ้าหยุดอยู่แล้ว ไม่ต้องหยุดซ้ำ

    if (direction != motion.Direction)
        motion.ToPreciseTarget(); // ปรับตำแหน่งให้พอดีศูนย์กลาง (กรณี FPS ต่ำๆ)

    CreateMotion(player.Position, direction);
}

1 reference
private void UnstableMoveOpposite(Vector2 direction)
{
    CreateMotion(motion.TargetPosition, direction);
}
2 references
private void CreateMotion(Vector2 oldPosition, Vector2 direction)
{
    var targetPosition = tileMap.TileCenter(oldPosition, direction);
    motion = new LinearMotion(player, speed: 300, targetPosition, direction);
}
```

ตรวจสอบการชน และการออกนอกขอบเขต

ไม่จำเป็นมากนัก

โจทย์

- ออกแบบ Tile Map ของตัวเองขนาด 6*6 ขึ้นไป
- ลองทำทางเดินแคบๆ เพื่อทดสอบการกดเลี้ยว
    - ทดลองกดเลี้ยวซ้ายขวาล่วงหน้าแต่เนิ่นๆ