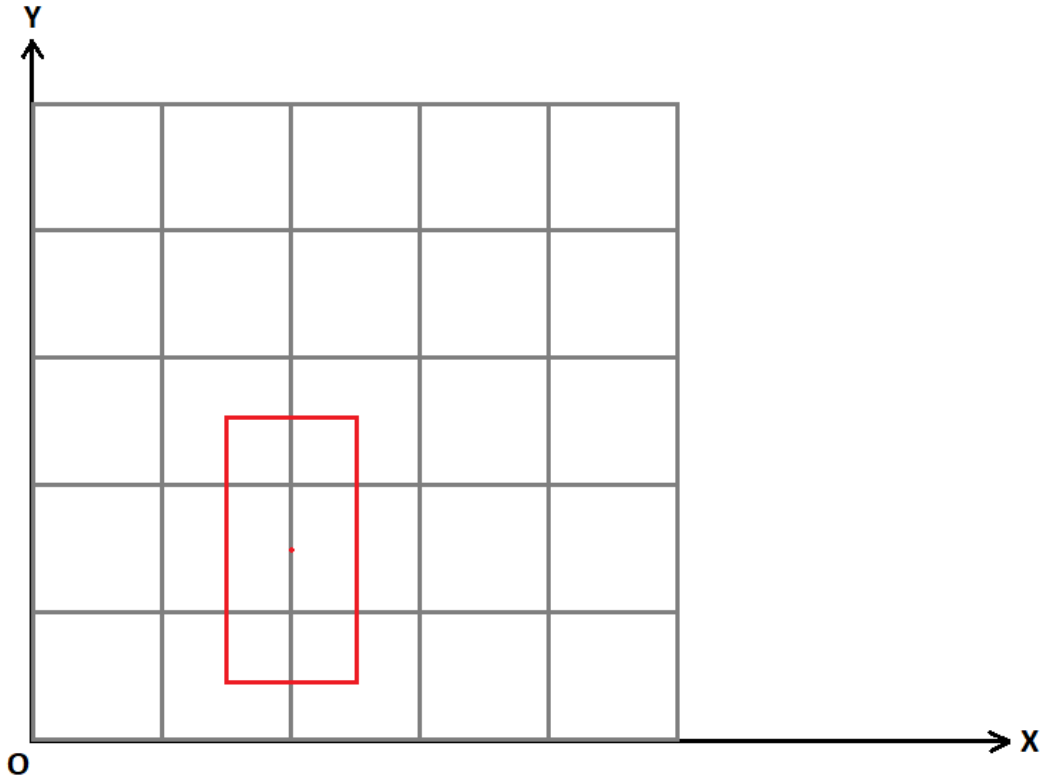
# CSD1130
# Game Implementation Techniques

QUIZ

TILEMAP-GRID INTERSECTION

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

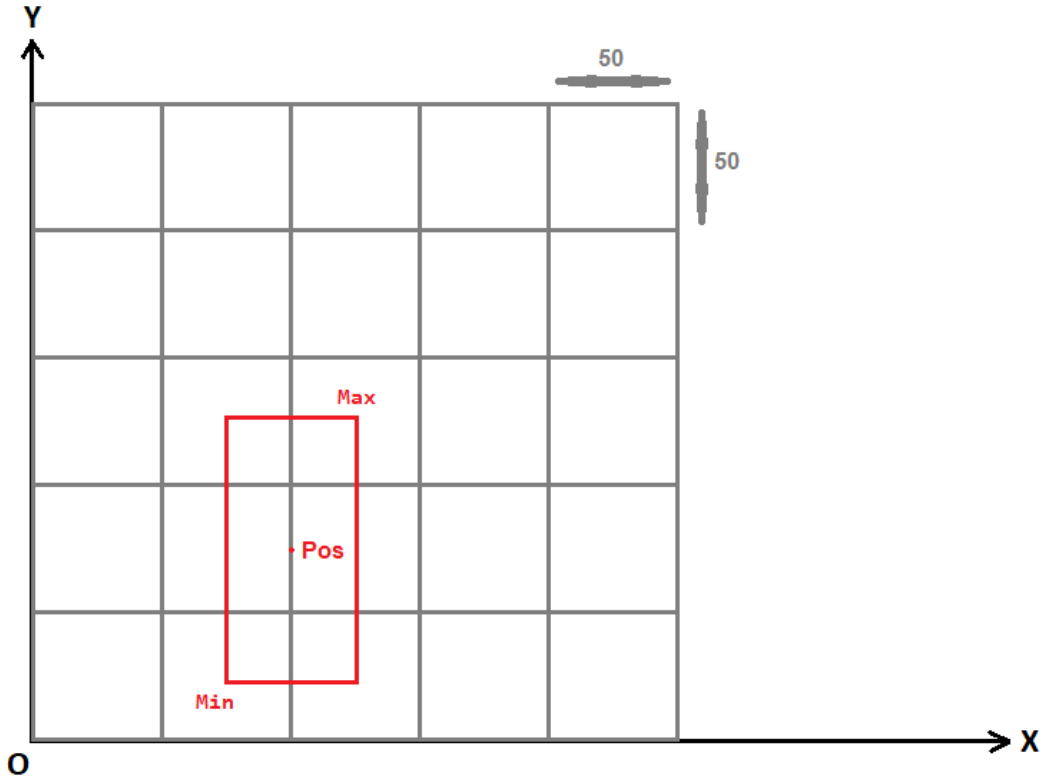
Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)




Question:
How would you return the cells that the game object is colliding with?

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

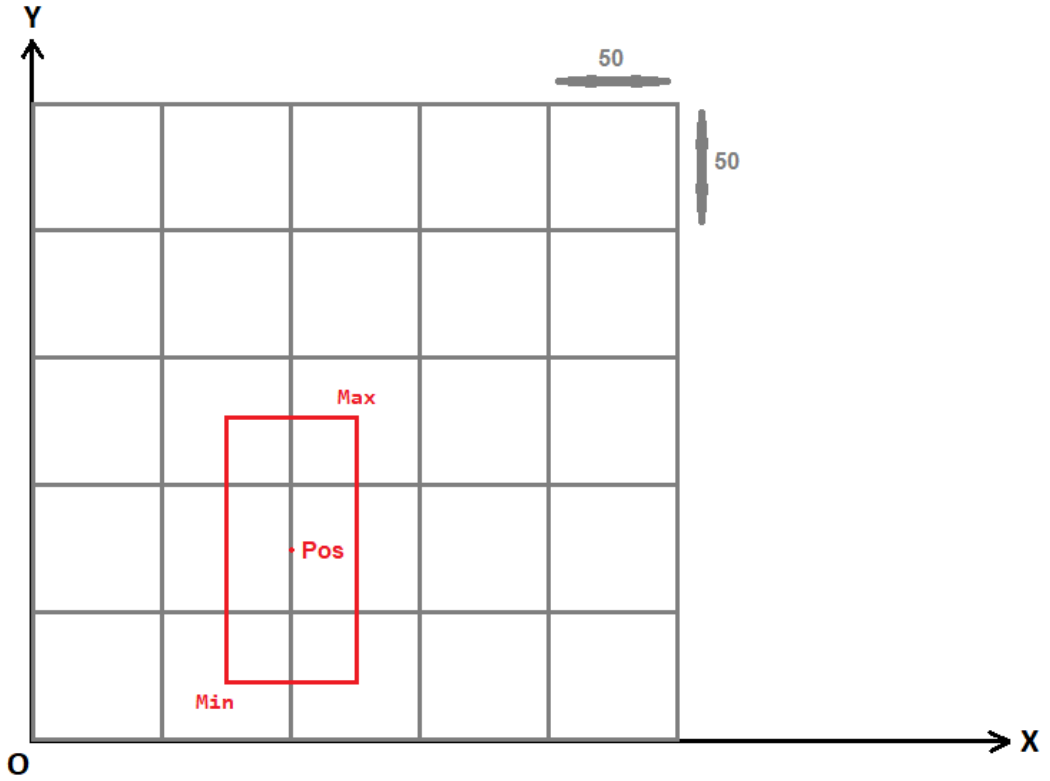
Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)




Question:
How would you return the cells that the game object is colliding with?

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)



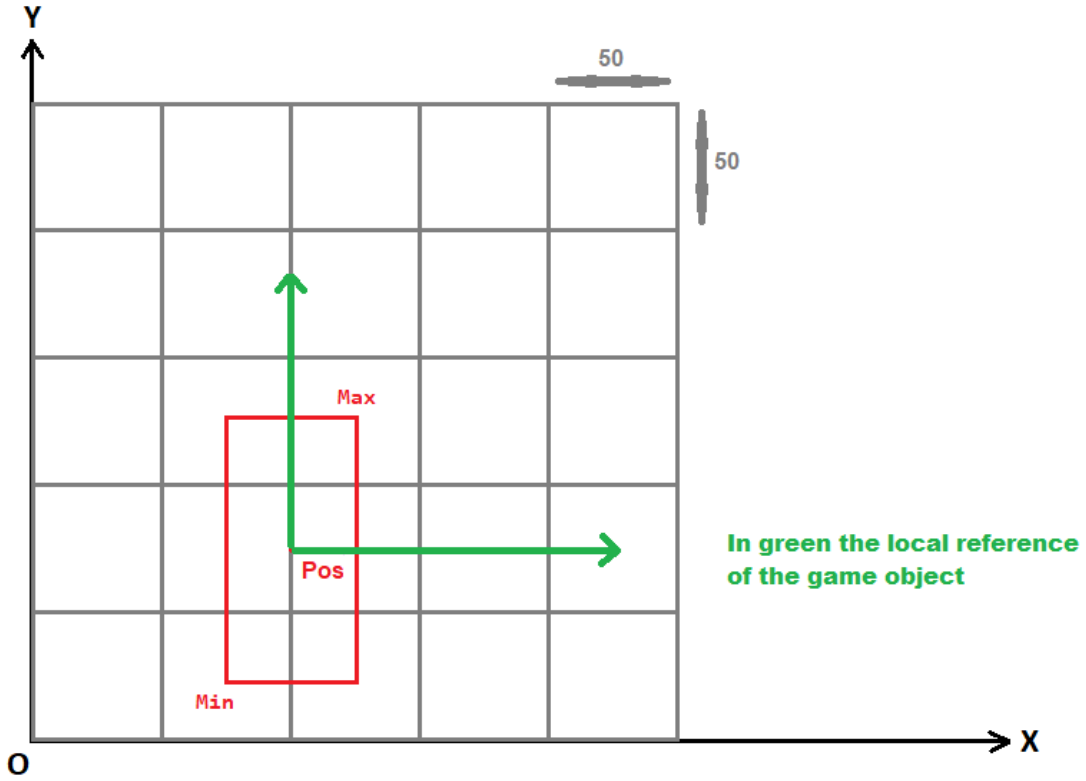

**Question:**
**How would you return the cells that the game object is colliding with?**

Answer:

First, find the cells where Min and Max reside. After that,you can conclude all the intermediate cells!

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)




**Question:**
**How would you return the cells that the game object is colliding with?**

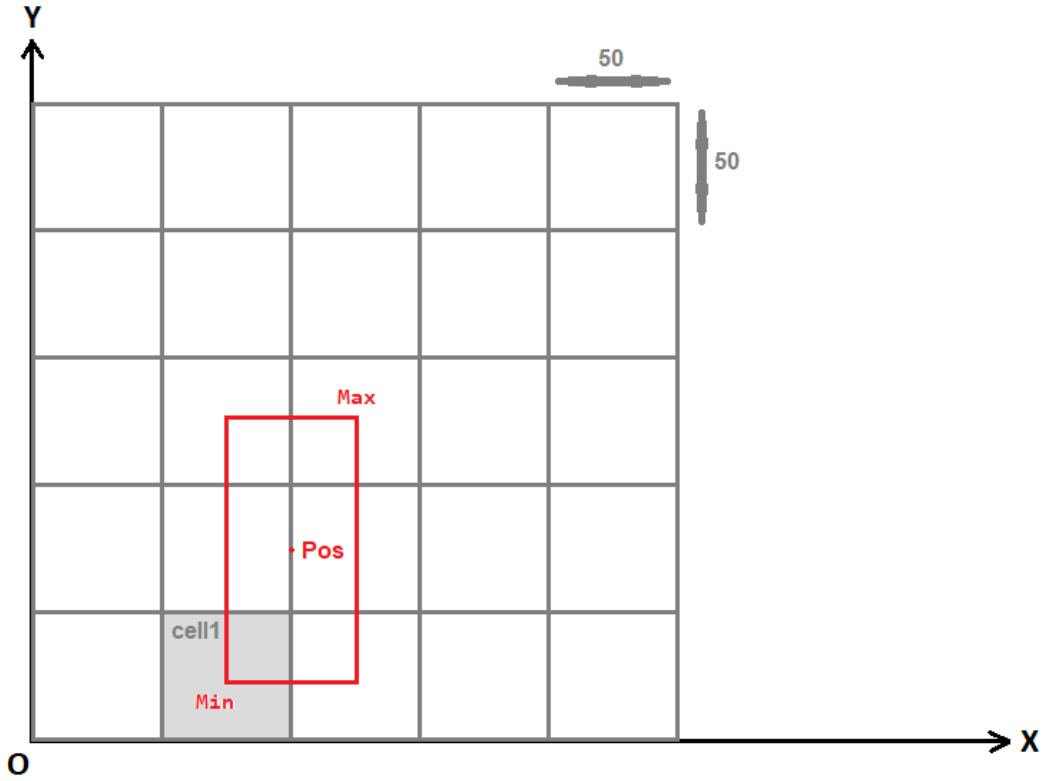
Answer:

Min and Max can be pre-calculated (load time), relative to the local origin reference of the Game Object.

e.g.
Min.X = -50 / 2
Min.Y = -100 / 2

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)




**Question:**
**How would you return the cells that the game object is colliding with?**

Answer:

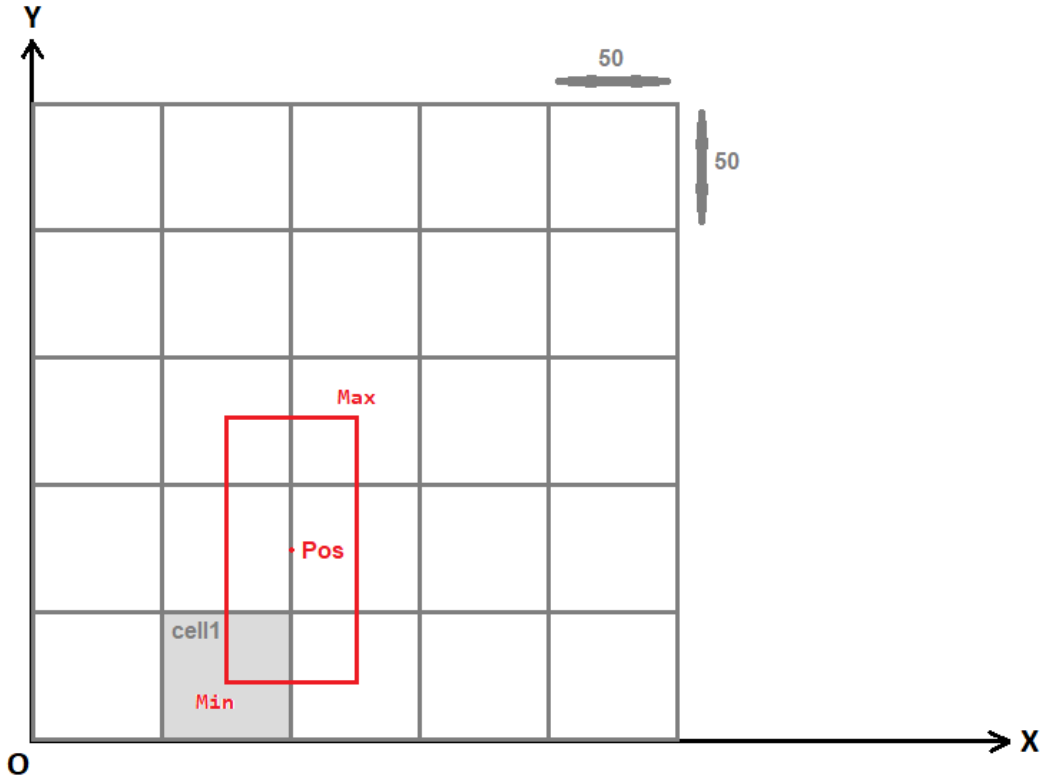
We need to divide the world coordinates of Min and Max onto the width and the height of a cell.

```
cell1_index.X = (GameObject.Pos.X + Min.X) / CELL_WIDTH;
cell1_index.Y = (GameObject.Pos.Y + Min.Y) / CELL_HEIGHT;

cell1_index.X = (int)cell1_index.X
cell1_index.Y = (int)cell1_index.Y
```

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)



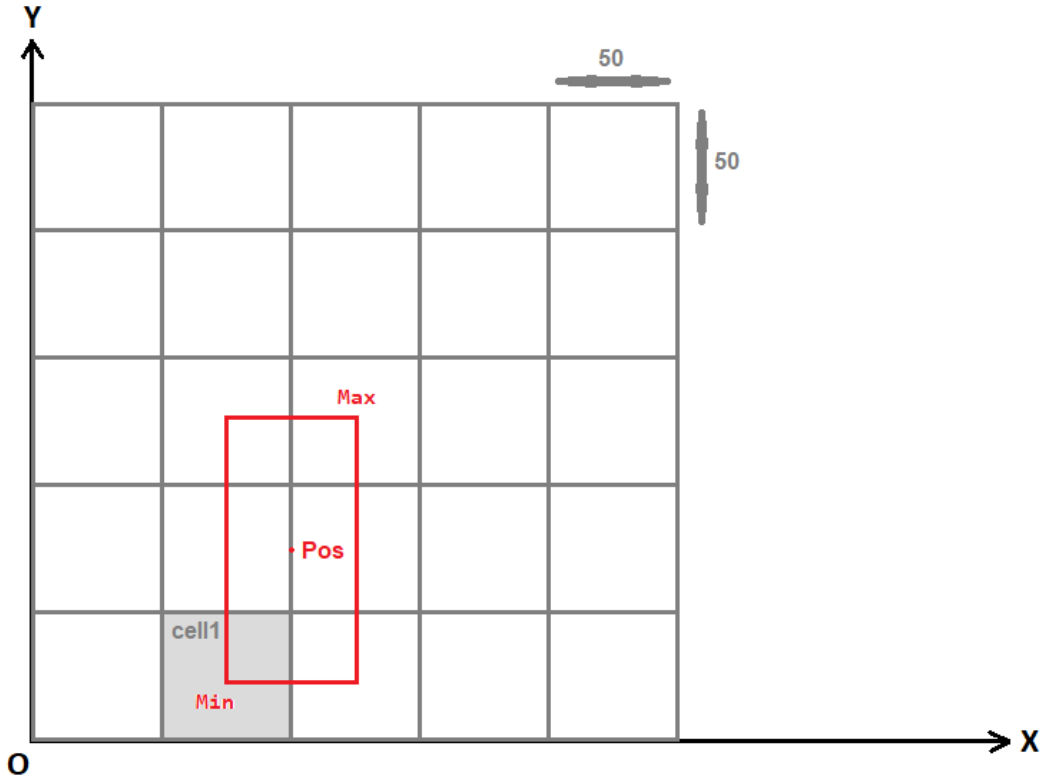

**Question:**
**How would you return the cells that the game object is colliding with?**

Answer:

This implies:
cell1_index(1,0) and cell2_index(2,2)
Now we can conclude all the intermediate cells.

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (50x50)

Game object: position: Pos(100, 75)
bounding rectangle(Width=50, Height=100)

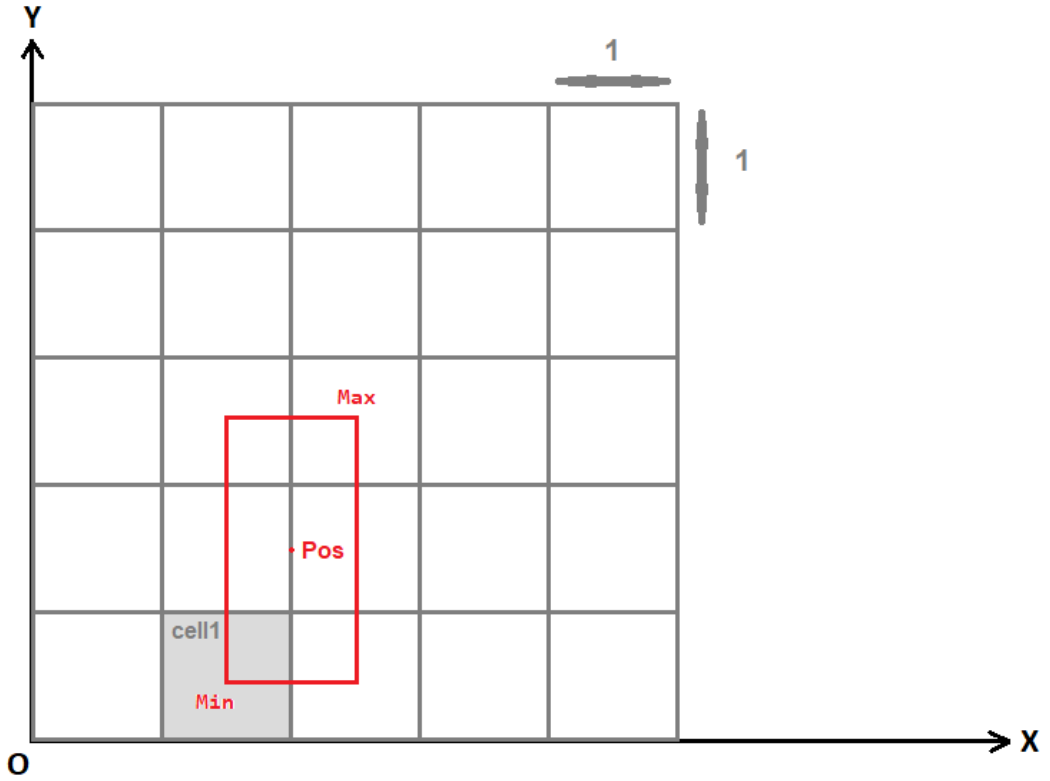



Question to think about:

What if our grid (tilemap), Game Object position and its bounding rectangle were all in a normalized system?

We have a grid (tilemap): position(0, 0)
Cell TM[5][5]
where a Cell has dimensions (1x1)

Game object: position: Pos(2.0, 1.5)
bounding rectangle (Width=1, Height = 2)

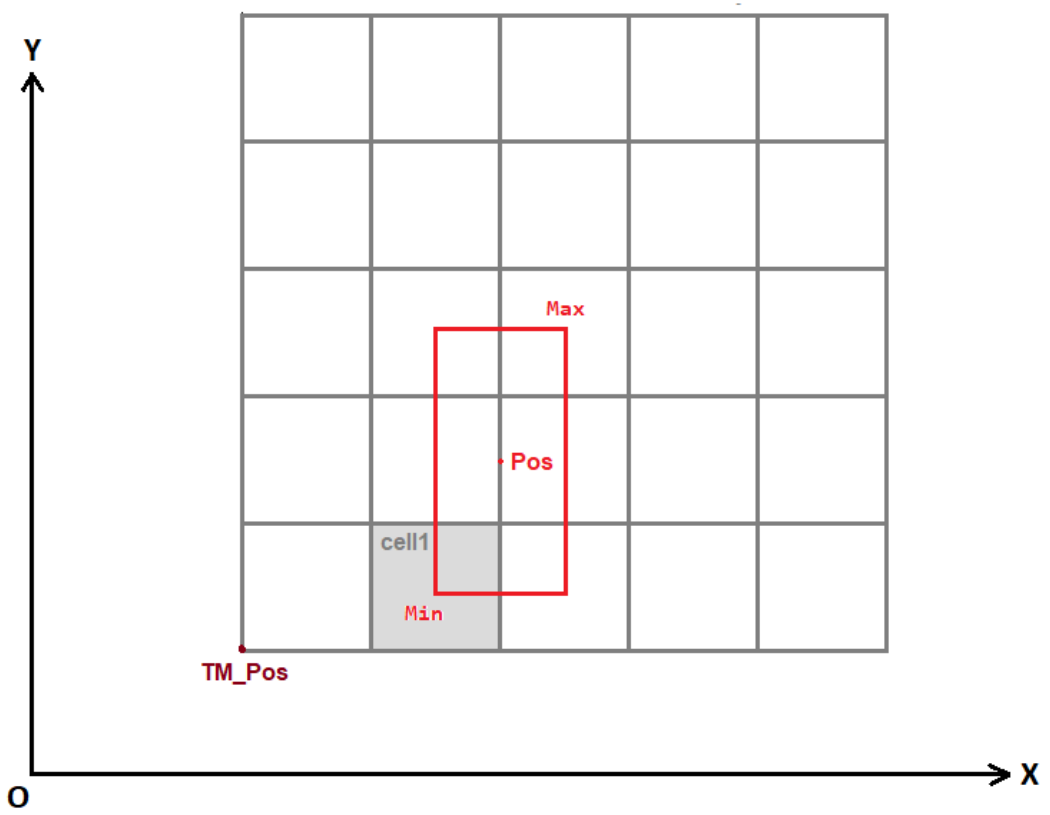



**Question to think about:**

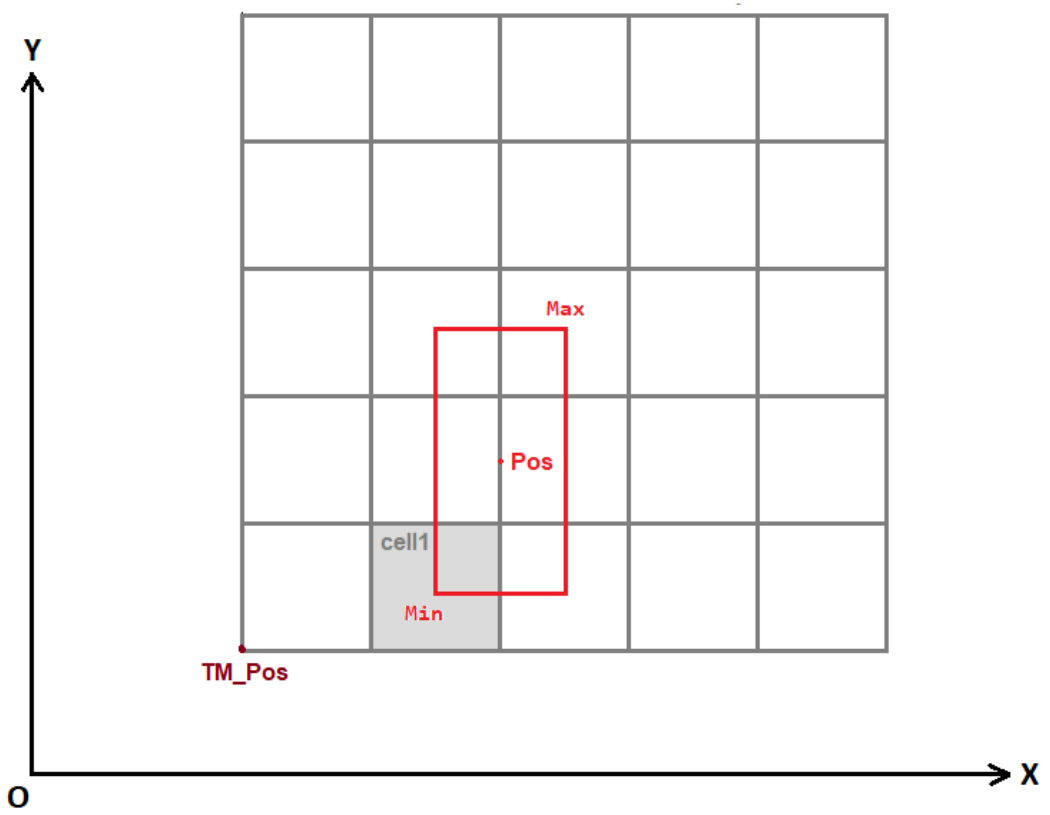
**What if our grid (tilemap), Game Object position and its bounding rectangle were all in a normalized system?**

```
cell1_index.X = (GameObject.Pos.X + Min.X);   //No need to divide by cell width!
cell1_index.Y = (GameObject.Pos.Y + Min.Y);   //No need to divide by cell height!
cell1_index.X = (int)cell1_index.X
cell1_index.Y = (int)cell1_index.Y
```

## What if the grid (tilemap) is not located at the origin of the world coordinates?



## What if the grid (tilemap) is not located at the origin of the world coordinates?




Answer:

```
cell1_index.X = (GameObject.Pos.X - TM_Pos.X + Min.X) / CELL_WIDTH;
cell1_index.Y = (GameObject.Pos.Y - TM_Pos.Y + Min.Y) / CELL_HEIGHT;

cell1_index.X = (int)cell1_index.X;
cell1_index.Y = (int)cell1_index.Y;
```